VISUALIZACIÓN GRÁFICA EN 3D DEL MOVIMIENTO CAPTURADO DE UNA PINZA MECÁNICA

EIVAR ARLEX ROJAS CASTRO

UNIVERSIDAD AUTONOMA DE OCCIDENTE

FACULTAD DE INGENIERÍA

DEPARTAMENTO DE AUTOMÁTICA Y ELECTRÓNICA

PROGRAMA DE INGENIERÍA ELECTRÓNICA

SANTIAGO DE CALI

2006

VISUALIZACIÓN GRÁFICA EN 3D DEL MOVIMIENTO CAPTURADO DE UNA PINZA MECÁNICA

EIVAR ARLEX ROJAS CASTRO

Trabajo de grado para optar al titulo de Ingeniero Electrónico

Director:
Msc. JESÚS DAVID CARDONA QUIROZ
Ingeniero de Producción

UNIVERSIDAD AUTONOMA DE OCCIDENTE
FACULTAD DE INGENIERÍA
DEPARTAMENTO DE AUTOMÁTICA Y ELECTRÓNICA
POGRAMA DE INGENIERÍA ELECTRÓNICA
SANTIAGO DE CALI
2006

Nota de aceptación:

Trabajo aprobado por el comité de grado en cumplimiento de los requisitos exigidos por la Universidad Autónoma de Occidente para optar al título de Ingeniero Electrónico.

DIEGO MARTINEZ CASTRO
Jurado

Santiago de Cali 19, Diciembre, 2006

Dedico este logro a toda mi familia, su apoyo incondicional siempre fue motivo para seguir adelante, gracias a mis padres Jaime y Geno, el gran soporte de mi vida y Esmeralda por estar siempre a mi lado.

AGRADECIMIENTOS

En agradecimiento por la colaboración de mis compañeros y amigos de universidad, Andrés, Edwin y Mario.

CONTENIDO

	Pág.
GLOSARIO	**176**
RESUMEN	**17**
INTRODUCCIÓN	**19**
1. OBJETIVOS	**22**
1.1. OBJETIVO GENERAL	**22**
1.2. OBJETIVOS ESPECÍFICOS	**22**
2. METODOLOGÍA	**24**
3. ESBOZO DE LA SOLUCIÓN	**26**
3.1. HARDWARE	**26**
3.2. SOFTWARE	**27**
4. SISTEMAS DE CAPTURA DE MOVIMIENTO	**28**
4.1. ESTRUCTURA	**31**
4.2. CLASIFICACIÓN	**35**
4.3. TECNOLOGÍA	**35**
4.3.1. Sistemas Acústicos.	**35**
4.3.2. Sistemas Mecánicos.	**36**

4.3.3. Sistemas Ópticos. 36

4.3.4. Sistemas Magnéticos. 38

4.4. FUENTE EMISOR-RECEPTOR 40

4.4.1. Sistema Inside-in. 40

4.4.2. Sistema Inside-out Magnéticos. 40

4.4.3. Sistema Outside-in. 41

4.5. ADQUISICIÓN DIRECTA - INDIRECTA 42

4.6. COMPARACION DE LOS SISTEMAS DE CAPTURA 43

5. ELECCIÓN DEL SISTEMA HARDWARE 44

5.1. SENSORES 44

5.2. SISTEMA DE ADQUISICIÓN DE DATOS 49

6. INTRODUCCIÓN A LAS LIBRERÍAS GRÁFICAS 51

6.1. LIBRERÍAS GRÁFICAS 3D 55

6.2. OPENGL 56

6.2.1. OpenGl Pipeline. 57

6.3. DIRECT3D 60

6.3.1. Direct3D Pipeline. 60

6.4. OPENGL Vs. DIRECT3D 64

6.4.1. Facilidad de Uso. 64

6.4.2. Portabilidad. 64

6.4.3. Extensiones. 65

6.4.4. Tabla de comparación. 66

6.4.5. Implementaciones. 68

7. DESARROLLO DEL SISTEMA SOFTWARE (REQUERIMIENTOS) 69

7.1. OBJETIVO GENERAL 69

7.2. OBJETIVOS ESPECIFICOS 69

7.3. ALCANCE 70

7.4. DEFINICIÓN DEL SISTEMA 71

7.5. ESPECIFICACIONES SUPLEMENTARIAS (NO FUNCIONALES) 72

7.6. DEFINICIÓN DE ACTOR 73

7.7. DIAGRAMA DE CASOS DE USO 74

8. DESARROLLO DEL SISTEMA SOFTWARE (ANÁLISIS) 75

9. DESARROLLO DEL SISTEMA SOFTWARE (DISEÑO) 79

9.1. REALIZACIÓN DE CASOS DE USO DISEÑO 79

9.2. PAQUETES DE DISEÑO 84

9.3. DESCRIPCION DE PAQUETES DE DISEÑO 85

10. DESARROLLO DEL PROTOTIPO 89

10.1. DESCRIPCIÓN 90

11. CONCLUSIONES **96**

12. TRABAJO FUTURO **97**

BIBLIOGRAFÍA **98**

ANEXOS **100**

LISTA DE ILUSTRACIONES

Pág.

Ilustración **1.** Estructura del prototipo de captura de movimiento 32
Ilustración **2**. Ubicación de los sensores en el brazo. 45
Ilustración **3**. Dimensiones y comportamiento de la resistencia del sensor 47
Ilustración **4**. Diagrama electrónico para el acondicionamiento de la señal generada por el sensor 48
Ilustración **5**. Diagrama en bloques de un sistema de adquisición de datos 49
Ilustración **6**. Diagrama interno del sistema de adquisición de datos 50
Ilustración **7**. Diagrama interno del Pipeline de OpenGl 58
Ilustración **8**. Diagrama interno del Pipeline de Direct3D 61
Ilustración **9**. Usuario de Sistema 73
Ilustración **10**. Diagrama de Casos de Uso 74
Ilustración **11**. Caso de Uso-Análisis 02 “Generar Captura de datos” 76
Ilustración **12**. Caso de Uso-Análisis 02 “Generar Captura de datos” Flujo Alternativo 77
Ilustración **13**. Caso de Uso-Análisis 02 “Generar Captura de datos” Flujo Alternativo 78
Ilustración **14**. Caso de Uso-Diseño 02 “Generar Captura de datos” (Diagrama de Clases) 80
Ilustración **15**. Caso de Uso-Diseño 02 “Generar Captura de datos” (Diagrama de Interacción) 81
Ilustración **16**. Caso de Uso-Diseño 02 “Generar Captura de datos” Flujo Alternativo (Diagrama de Interacción) 82
Ilustración **17**. Caso de Uso-Diseño 02 “Generar Captura de datos” Flujo Alternativo (Diagrama de Interacción) 83
Ilustración **18**. Paquetes de diseño 84

Ilustración **19**. Sensores flexibles 90
Ilustración **20**. Diagrama esquemático del circuito seguidor de voltaje. 91
Ilustración **21**. Modelo tridimensional 93
Ilustración **22**. Herramienta Skin 93
Ilustración **23**. Herramienta para exportar 95
Ilustración **24**. Imágenes del Prototipo de captura 95
Ilustración **25**. Caso de Uso-Requisitos 01 “Calibrar y Configurar el sistema Hardware de Captura” 106
Ilustración **26**. Caso de Uso-Requisitos 02 “Generar Captura de Datos” 108
Ilustración **27**. Interfaces de la aplicación 109
Ilustración **28**. Caso de Uso-Análisis 01 “Calibrar Sistema Hardware de Captura” 111
Ilustración **29**. Caso de Uso-Análisis 01 “Calibrar Sistema Hardware de Captura” 112
Ilustración **30**. Caso de Uso-Análisis 01 “Calibrar Sistema Hardware de Captura” (Flujo Alternativo) 113
Ilustración **31**. Caso de Uso-Análisis 03 “Manipulación visor 3D” 114
Ilustración **32**. Caso de Uso-Análisis 04 “Reproducir Captura de movimiento” 115
Ilustración **33**. Caso de Uso-Análisis 04 “Reproducir Captura de movimiento” Flujo Alternativo 116
Ilustración **34**. Caso de Uso-Diseño 01 “Calibrar Sistema Hardware de Captura” (Diagrama de Clases) 117
Ilustración **35**. Caso de Uso-Diseño 01 “Calibrar Sistema Hardware de Captura” (Diagrama de Interacción) 118
Ilustración **36**. Caso de Uso-Diseño 01 “Calibrar Sistema Hardware de Captura” Flujo Alternativo (Diagrama de Interacción) 119
Ilustración **37**. Caso de Uso-Diseño 01 “Calibrar Sistema Hardware de Captura” Flujo Alternativo (Diagrama de Interacción) 120

Ilustración **38**. Caso de Uso-Diseño 03 “Manipular el Visor 3D” (Diagrama de Clases) 121
Ilustración **39**. Caso de Uso-Diseño 03 “Manipular el Visor 3D” Flujo Alternativo (Diagrama de Interacción) 122
Ilustración **40**. Caso de Uso-Diseño 04 “Reproducir Captura de Movimiento” (Diagrama de Clases) 123
Ilustración **41**. Caso de Uso-Diseño 04 “Reproducir Captura de Movimiento” (Diagrama de Interacción) 124
Ilustración **42**. Caso de Uso-Diseño 04 “Reproducir Captura de Movimiento” Flujo Alternativo (Diagrama de Interacción) 125
Ilustración **43**. Vértices que conforman un triángulo (v1 (1, 2, 2).v2 (2, 5, 4).v3 (5, 1, 0).) 127
Ilustración **44**. Composición de un cubo por 12 triángulos 129
Ilustración **45**. Normales de una geometría 130
Ilustración **46**. Lista de vértices 130
Ilustración **47**. Lista de líneas 130
Ilustración **48**. Tira de líneas 131
Ilustración **49**. Lista de triángulos 131
Ilustración **50**. Tira de triángulos 131
Ilustración **51**. Abanico de triángulos 132
Ilustración **52**. Pipeline Direct3D 133
Ilustración **53**. Coordenadas de transformación de modelo 134
Ilustración **54**. Coordenadas de transformación de cámara 135
Ilustración **55**. Planos de corte 135
Ilustración **56**. Librerías de referencia. 136
Ilustración **57**. Creación y dibujado en el canvas. 141
Ilustración **58**. Dibujado de un triángulo en el canvas 145
Ilustración **59**. Orden de construcción de un triángulo 146

Ilustración **60**. Orden de construcción de un cubo 147
Ilustración **61**. Dibujado de un cubo en el Canvas 155
Ilustración **62**. Coordenadas de mapa 157
Ilustración **63**. Ejemplo de textura 157
Ilustración **64**. Dibujado de un cubo con textura 158
Ilustración **65**. Eje de coordenadas locales 160
Ilustración **66**. Transformación de la posición y rotación de un objeto de forma incorrecta 160

LISTA DE TABLAS

Pág.

Tabla **1.** Comparación sistemas de captura 43

Tabla **2.** Tabla de comparación de OpenGl y Directx. 66

LISTA DE ANEXOS

	Pág.
Anexo 1. Descripción de casos de uso	100
Anexo 2. Modelos de arquitectura	106
Anexo 3. Desarrollo del sistema software (Análisis)	111
Anexo 4. Desarrollo del sistema software (Diseño)	117
Anexo 5. Tutorial Direct3D	126

GLOSARIO

ALPHA CHANNEL: en el formato de representación de imágenes dividiéndolos en los colores primarios Rojo, Verde y Azul (canales RGB) se añade un cuarto canal (alpha) el cual indica la cantidad de transparencia tiene la imagen respecto al fondo utilizado para mostrarla.

BUMP: textura o imagen utilizada para modificar la normal de reflexión de la fuente de luz sobre un objeto, dando la apariencia de rugosidades.

FRAME: cuadro o imagen individual de las que componen una animación. La sucesión de Frames consecutivos con pequeñas diferencias producen la ilusión de movimiento. La velocidad de la animación se mide en Frames por segundo o fr/s.

MESH: término que se refiere a una figura en 3D, en general que esté formada por polígonos.

RENDER: término que se refiere a la generación de un FRAME o imagen individual. El render consume potencia de cómputo debido a los cálculos necesarios segun el número de objetos, luces y efectos en la escena y segun el punto de vista de la cámara.

TEXTURA: imagen proyectada sobre parte o la totalidad de la superficie de un objeto para darle apariencia fotorrealistica. Existen texturas procedimentales (calculadas) o texturas de mapa de imagen.

WIREFRAME: forma de presentación de los objetos 3D en pantalla por medio de 'alambres' o líneas que representan el volumen.

RESUMEN

En este trabajo se presenta el proceso de diseño de un sistema de captura de movimiento para un brazo humano, el cual consta de dos etapas: Hardware y Software.

El Hardware es el encargado de medir los cambios en los puntos de inflexión y la transmisión de las lecturas al computador. Estará dividido en los siguientes subsistemas:

- Sensores: Existen 8 puntos de inflexión de interés para el sistema de captura. Estos puntos describen en su totalidad los movimientos principales que puede ejecutar el brazo y la mano. El objetivo de estos sensores es determinar el ángulo absoluto que efectúa cada punto de inflexión. Su comportamiento debe ser completamente lineal y exacto. Además, debe permitir el libre movimiento de cada punto que se desea medir.

- Acondicionamiento: Este subsistema recibe la señal de voltaje variable que generan el grupo de sensores. Debe filtrar los ruidos no deseados que se sumen a la señal y permitir ajustar el nivel de voltaje de cada sensor. Esto para efectos de que un sensor pierda sus propiedades iniciales de medición.

- Captura y envío*:* Esta etapa se encarga de ser la interfaz entre el computador y los sensores. Cada sensor ingresa su señal por un canal separado de lectura. Este subsistema debe tener la capacidad de recibir instrucciones de lectura desde el ordenador, tanto para un sensor como para el grupo de los 8 dispositivos.

Este sistema software realiza la gestión de los datos capturados, su representación y la interacción con el entorno de visualización. Contiene las siguientes partes de desarrollo:

- Calibración del Hardware: Esta parte con la ayuda del usuario, ajustará los valores de conversión para lecturas realizadas por el sistema hardware. Además, determinar el estado de conexión que se encuentra cada sensor.

- Captura de Movimiento: Encargado de sincronizar la lectura realizada por el sistema Hardware y la representación tridimensional de los movimientos. Cada valor capturado los sensores será almacenado en un archivo, que en futuro puede ser visualizado en el sistema software.

- Representación tridimensional: Permite dotar al sistema de captura la visualización de los datos capturados por el sistema hardware. Estará dotado de un brazo tridimensional, el cual estará dotado de un sistema jerárquico de huesos que manipularan su geometría, replicando los movimientos efectuados por el brazo.

INTRODUCCIÓN

En este trabajo se presenta el proceso de diseño de un sistema de captura de movimiento para un brazo humano, el cual consta de dos etapas: Hardware y Software.

El desarrollo esta enmarcado por la creación de un sistema de visualización tridimensional donde además de visualizar un brazo en tres dimensiones se logre manipular su estructura geométrica. El segundo, obtener los cambios generados por los puntos de inflexión del brazo mediante un grupo de sensores que generan señales de voltaje que se convierten en valores digitales y enviados al computador. Cada sensor estará dispuesto a lo largo del brazo, muñeca, y dedos de la extremidad derecha.

RAZONES DEL PROYECTO

La motivación para el desarrollo de este proyecto inició en 1999, con la creación del Grupo de investigación de Visualización y Simulación 3D donde su principal objetivo se basó alrededor de los gráficos tridimensionales.

Durante 3 años, se realizó una investigación general sobre la generación de imágenes tridimensionales. Una vez recolectada la información necesaria, se intentó generar un aeroplano a partir de planos, para esto, se utilizó una herramienta CAD. La herramienta utilizada contaba con limitaciones de representación tridimensional, fundamentalmente una representación de tipo Ortogonal, lenta, sumada a un sistema de asignación de materiales muy básica, arrojando como consecuencia la generación de imágenes poco realistas.

Tras la investigación de varias herramientas, se optó por utilizar una herramienta mucho más especializada en la generación de escenas tridimensionales: 3D Studio Max versión 2.5, durante aproximadamente un año, se aprendió a utilizar la herramienta logrando obtener resultados muy satisfactorios. Posteriormente se estudiaron otras herramientas generadoras de contenido 3D, como Maya, LightWave y Rhinoceros. Pero como conclusión, todos estos sistemas generan como únicas salidas imágenes estáticas o videos, estos últimos sujetos a la animación previa de una cámara.

A raíz de la investigación continua del grupo, se descubrió que existían API´S gráficas generadoras de contenido 3D en tiempo real, y una de estas la API gráfica DirectX versión 6.1.

El primer tipo de experiencia que se generó con esta API gráfica, fue visualizar un objeto con volumen en tiempo real. A medida que se conocía mucho más esta herramienta se generaron aplicaciones más complejas. Una de estas aplicaciones lograba visualizar el movimiento de una máquina posicionadora XY en tiempo real, cuando la máquina se desplazaba en cualquier sentido los sensores generaban una señal eléctrica, esta señal se convertía en datos binarios los cuales eran recibidos por un computador e interpretados por la aplicación entregando como salida el movimiento del la máquina posicionadora XY en tres dimensiones sobre la pantalla del computador. Hasta este punto lo único que se realizaban eran movimientos de objetos compuestos y nunca su geometría fue afectada.

El interés por el continuo desarrollo de contenido tridimensional en tiempo real generó la necesidad de crear un sistema que lograra generar deformaciones a un objeto tridimensional. Estas deformaciones son generadas en el mundo real y transmitidas al computador.

En su mayoría, los diseños de sistemas de captura de movimiento se enfocan a la generación de dispositivos hardware, dejando en un segundo plano el desarrollo de los entornos tridimensionales de visualización. El propósito de este proyecto se orienta además del diseño y construcción de un sistema hardware, al diseño y construcción de un sistema Software de visualización 3D.

1. OBJETIVOS

1.1. OBJETIVO GENERAL

Desarrollar un sistema de captura de movimiento capaz de interpretar, convertir, representar y almacenar los movimientos generados por un brazo humano.

1.2. OBJETIVOS ESPECÍFICOS

- Enfocar el desarrollo del sistema Software en la utilización de una librería gráfica tridimensional.

- Dotar al visualizador tridimensional de herramientas que facilite la visualización del entorno permitiendo funcionalidades como alejamiento o acercamiento, desplazamiento de la pantalla y rotación del entorno (Zoom, Pan, TrackBall).

- Incluir un brazo tridimensional que responda a las señales generadas por el sistema Hardware encargada de la captura de movimiento.

- Permitir el almacenamiento en disco de datos capturados por el sistema hardware.

- Permitir la reproducción de datos almacenados en disco de la captura y su visualización a través del brazo tridimensional.

- Desarrollar un sistema de sensores capaces de describir el movimiento del brazo humano.

- Implementar un sistema de adquisición de datos para la captura de los movimientos.

2. METODOLOGÍA

La metodología del desarrollo de este sistema iniciará con la identificación de los dos sistemas que componen el proyecto Hardware y Software. Definiendo los aspectos afines con el desarrollo de cada sistema.

Una vez concluida la planificación se inicia el proceso de recopilación, análisis y síntesis, de la base teórica para el desarrollo del proyecto. Está descrita por los siguientes pasos:

- Investigación de dispositivos para la captura de los valores generados por los puntos de inflexión, en este paso se recopilará información sobre los dispositivos y lo métodos para capturar el ángulo y (o) posición de un objeto. Debido a que el objetivo es determinar los ángulos generados por los puntos de inflexión presentes en el brazo.

- Elección del sistema de adquisición de datos, se debe determinar cual es la mejor razón costo-beneficio para realizar captura de datos del mudo real. Enfocando la investigación a sistemas de alta velocidad y practicidad.

- Manejo de gráficos tridimensionales, por los antecedentes previamente descritos el desarrollo de esta etapa estará centrado en la API Gráfica Directx versión 9c, el análisis estará orientado a la generación de gráficos en tiempo real.

- Sistema jerárquico de huesos, establecer el sistema que manipulará la forma geométrica del brazo tridimensional.

Obteniendo las bases teóricas necesarias para el desarrollo del sistema de captura. Se inicia la etapa del sistema Software, la metodología para su desarrollo consta de las siguientes etapas: Recopilación de requisitos, Análisis, Diseño, Implementación.

Concluidas las dos etapas esenciales del proyecto se integran y se realizan las pruebas de los tiempos de captura y representación 3D, determinando el tiempo máximo que debe tomar el sistema hardware para la captura de datos de los sensores y el tiempo necesario para la transformación y representación del modelo 3D del brazo.

Al establecer los tiempos mínimos y máximos de captura de datos, se determina el plazo máximo durante el cual se puede realizar una captura de movimiento evitando que el sistema se bloquee o que se quede sin recursos de memoria.

Concluyendo la fase de pruebas se ajusta el sistema a las condiciones ideales de captura, para lograr el máximo desempeño.

3. ESBOZO DE LA SOLUCIÓN

3.1. HARDWARE

Este bloque será el encargado de medir los cambios en los puntos de inflexión y la transmisión de las lecturas al computador. Estará dividido en los siguientes subsistemas:

- Sensores: Existen 8 puntos de inflexión de interés para el sistema de captura. Estos puntos describen en su totalidad los movimientos principales que pueden ejecutar el brazo y la mano. El objetivo de estos sensores es determinar el ángulo absoluto que efectúa cada punto de inflexión.

- Su comportamiento debe ser completamente lineal y exacto. Además, debe permitir el libre movimiento de cada punto que se desea medir.

- Acondicionamiento: Este subsistema recibe la señal de voltaje variable que generan el grupo de sensores. Debe filtrar los ruidos no deseados que se sumen a la señal y permitir ajustar el nivel de voltaje de cada sensor. Esto para efectos de que un sensor pierda sus propiedades iniciales de medición.

- Captura y envío: Esta etapa se encarga de ser la interfaz entre el computador y los sensores. Cada sensor ingresa su señal por un canal separado de lectura. Este subsistema debe tener la capacidad de recibir instrucciones de lectura desde el ordenador, tanto para un sensor como para el grupo de los 8 dispositivos.

3.2. SOFTWARE

Este sistema realizará la gestión de los datos capturados, su representación y la interacción con el entorno de visualización. Contiene las siguientes partes de desarrollo:

- Calibración del Hardware: Esta parte con la ayuda del usuario, ajustará los valores de conversión para lecturas realizadas por el sistema hardware. Además, determinar el estado de conexión que se encuentra cada sensor.

- Captura de Movimiento: Encargado de sincronizar la lectura realizada por el sistema Hardware y la representación tridimensional de los movimientos. Cada valor capturado los sensores será almacenado en un archivo, que en futuro puede ser visualizado en el sistema software.

- Representación tridimensional: Permite dotar al sistema de captura la visualización de los datos capturados por el sistema hardware. Estará dotado de un brazo tridimensional, el cual estará dotado de un sistema jerárquico de huesos que manipularan su geometría, replicando los movimientos efectuados por el brazo.

4. SISTEMAS DE CAPTURA DE MOVIMIENTO

La captura de movimiento para la animación de caracteres por computadora es relativamente nueva, comenzado a finales de los años 70 y solamente hasta ahora se esta expandiendo su uso a diferentes ramas.

La captura de movimiento es la capacidad de poder grabar los movimientos del cuerpo humano para su análisis, esto puede ser de forma inmediata o retardada. La captura de movimiento puede ser tan general como la posición de un simple cuerpo en el espacio o tan compleja como la representación de gestos y deformaciones de la cara. La captura puede ser directa, por ejemplo el movimiento del brazo humano controlando el movimiento de un brazo de un personaje en el computador, o indirecto cuando la captura se utiliza para modificar otros aspectos como color de la piel, estados de ánimo, etc.

La idea de replicar movimiento humano no es nada nuevo, el RotosCoping fue una idea utilizada por Disney para colocar personajes humanos animados fuera de escena en la película Blanca Nieves. Luego de esto en los estudios de gráficos en el Instituto Nueva York Rebeca Allen utilizó un espejo denominado Half-Silvered para sobreponer las videocintas de bailarines verdaderos sobre la pantalla del computador para producir un bailarín generado por computador. Este sistema no es nada automáticos puesto que los animadores tenían que refinar los movimientos para hacerlos mas reales.

Entre 1980 y 1983 el profesor Tom Calvert un profesor de Kinesiología en la universidad Simon Fraser con ayuda de unos potenciómetros unidos al cuerpo produjo señales las cuales fueron al computador y se utilizaron para el estudio clínico de anormalidades del movimiento. El sistema consistía en realizar una

especie de exoesqueleto, donde cada sensor iba en cada articulación, se producía una señal análoga la cual se convertía en digital y transmitida al computador.

Pronto después en 1982 en el laboratorio de gráficos por computador Instituto de New York se utilizó un sistema de cámaras para hacer seguimiento a una serie de marcadores, que eran pequeños LED que destellaban, la combinación Hardware-Software determinaba la posición de cada marcador en cada cámara y comparando las imágenes determinaba la posición aproximada del marcador.

En 1988 Degraf/Wahrman con ayuda de un simple titiritero pudieron controlar en tiempo real muchos gestos de la cara de un títere virtual, incluyendo boca ojos y expresiones, llamado “La Cabeza que Habla”. Silicon Graphics proporcionó interpolaciones en tiempo real entre la manipulación de los controles y la marioneta tridimensional.

Al mismo tiempo en los estudios de Jim Henson Productions con ayuda de Silicon Graphics 4D Workstation, realizaron un exoesqueleto que iba ubicado desde el maxilar superior hasta el maxilar inferior, el cual poseía 8 grados de libertad, capturaba todos los movimientos de la boca, los cuales se representaban en tiempo real por una marioneta en el computador.

Posteriormente PDI desarrolló un exoesqueleto que iba desde la cintura hacia arriaba, capturando el movimiento de el torso, los brazos y la cabeza. Este sistema fue utilizado en muchos proyectos aunque no pudieron implementar un sistema del cuerpo completo a cause de que los sensores y la naturaleza de estos generaban mucho ruido.

Los sistemas de Captura de Movimiento se pueden dividir en cuatro categorías:

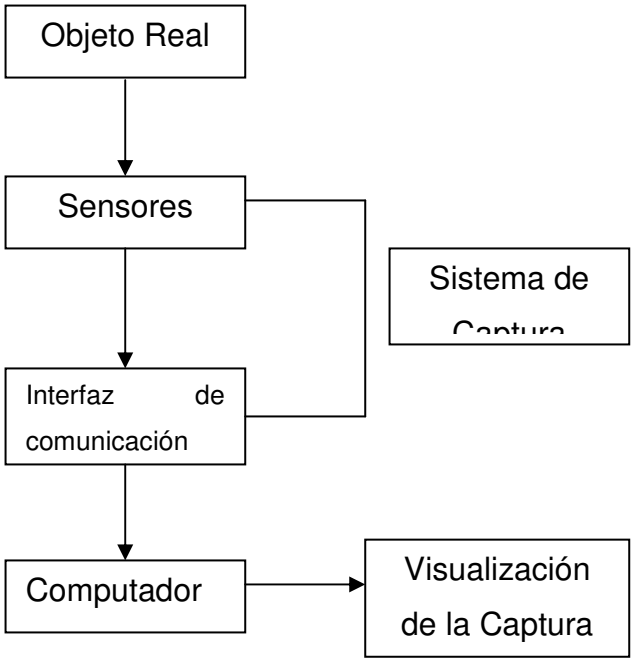
- Acústicos.
- Mecánicos.
- Ópticos.
- Magnéticos.

Actualmente, los sistemas mas utilizados son los basados en tecnologías ópticas y magnéticas.

4.1. ESTRUCTURA

Los sistemas de Captura de Movimiento son, por definición, sistemas que generan para un computador información que representan medidas físicas de movimiento capturado. Un proceso de captura consiste, de forma general en vestir un actor con un traje especial, donde serán posicionados reflectores (Sistemas Ópticos) o transmisores (Sistemas Magnéticos). Estos marcadores son generalmente posicionados en la llamada “Articulaciones Universales”, que son 19 posiciones que ofrecen un mínimo de precisión para representación de un movimiento humano [Blinn 87]. En sistemas mecánicos como Gipsy 3, de Meta Motion, un traje especial contiene diversos potenciómetros, también posicionados estratégicamente, que capturan las posiciones y orientaciones de las principales articulaciones. En general los sistemas de captura tienen la siguiente estructura:

Ilustración **1.** Estructura del prototipo de captura de movimiento



.

Algunas de las etapas del gráfico anterior pueden ser omitidas, dependiendo de la tecnología utilizada en el proceso. En 1993, Kalawsky [Kalawsky 93] propuso el siguiente conjunto de medidas para la especificación de sistemas de captura de movimiento humano:

Medidas Descriptivas:

- Proceso de captura a utilizar (Acústico, Electromagnético, Óptico o Mecánico).
- Ubicación de los sensores (Sensores en el Cuerpo o fuera de el).
- Parte del cuerpo a procesar (Espina dorsal, cabeza, hombros, brazos, piernas, dedos, iris).

Medidas Estáticas:

- Resolución espacial, alcance.
- Medida espacial.
- Linealidad, histéresis, calibración.

Medidas Dinámicas:

- Medida dinámica (Similar a medida estática).
- Resolución temporal, alcance de la frecuencia.
- Taza de muestreo, medida temporal, latencia.

Medidas de precisión:

- Estabilidad, Durabilidad.
- Ruido, interferencia externa, filtración.

Medidas de interfase:

- Interfaz con el sensor (Cámara, Transmisores) (Voltaje AC/DC, resistencia etc.).
- Interfaz de con el usuario.
- Interfaz de comunicación (RS232, RS422, IEEE488 etc.).

Medidas Computacionales:

- Ejes de referencia (ejes de coordenada local, ejes de coordenada global).

Medidas operacionales:

- Factor de forma (tamaño y peso de los sensores).
- Tiempo de preparación para la captura.
- Compatibilidad con otros sistemas.
- Ambiente de operación (Humedad, Temperatura).

4.2. CLASIFICACIÓN

Se pueden utilizar diversas formas de clasificación de los sistemas de captura, de acuerdo a la posición de la fuente emisora respecto a los sensores, en cuanto a la forma de la obtención de los datos capturados.

4.3. TECNOLOGÍA

Esta clasificación es la más importante al definir explícitamente el proceso utilizado para la captura de movimiento. Se pueden dividir en cuatro categorías: acústicos, mecánicos, ópticos y magnéticos. Los dos últimos son los más utilizados hoy en día por las industrias de entretenimiento.

4.3.1. Sistemas Acústicos. En este tipo de sistema, un conjunto de emisores sonoros son colocados en las principales articulaciones del actor, a su vez tres receptores recibirán la señal emitida por los sensores; dichos receptores no se encontrarán ubicados en la zona local de captura. Los transmisores son accionados secuencialmente para producir un ruido característico que será captado por los receptores, los cuales calcularán la poción en el espacio.

El cálculo de la posición de cada transmisor se realiza de la siguiente forma: utilizando el tiempo del recorrido que realiza el ruido emitido por el transmisor hasta llegar al receptor. Tomando en cuenta la velocidad del sonido se calcula la distancia entre el emisor y el receptor utilizando el efecto Dopler. Para la posición 3D de cada transmisor se realiza una triangulación con los datos arrojados por los tres receptores.

Uno de los problemas de este método es la dificultad de obtener información precisa en un instante de tiempo, debido al disparo secuencial de los transmisores. Además de esto, los sistemas acústicos sufren del mismo tipo de problema que los sistemas magnéticos, siendo incómodos y perjudicando la movilidad del actor reduciendo así la cantidad de movimientos que pueden ser ejecutados. El número de transmisores que pueden ser utilizados simultáneamente es limitado e impiden una descripción correcta del movimiento capturado.

4.3.2. Sistemas Mecánicos. Los sistemas mecánicos de captura no cuentan con una tecnología muy avanzada pero poseen algunas ventajas que resultan altamente atrayentes en la industria de la cinematografía.

Los sistemas mecánicos están compuestos de potenciómetros, posicionados en las articulaciones deseadas, permiten obtener las posiciones y orientaciones a tazas de muestreo muy altas (Tiempo real). Una de las ventajas de este tipo de sistema es que posee una interfaz parecida a las utilizadas en los sistemas de stop-motion.

Los sistemas mecánicos no son afectados por campos magnéticos o interferencias indeseadas, problemas típicos de los sistemas magnéticos y ópticos. Además no necesitan de un proceso largo de calibración.

4.3.3. Sistemas Ópticos. En este tipo de sistema, un actor viste un traje recubierto de reflectores (en general emisores LED), posicionados en las principales articulaciones. Cámaras especiales son ubicadas estratégicamente para hacer seguimiento de los reflectores durante el movimiento del actor. Cada cámara genera coordenadas 2D para cada reflector (obtenidas mediante un

proceso de segmentación). Con el conjunto de coordenadas 2D capturadas por las cámaras, se analizan mediante un software que integrara esta información transformándolas en coordenadas 3D.

Existen algunas variaciones entre diferentes sistemas ópticos de captura. El sistema MultiTrax, de Adaptive Optics Associates, contiene marcadores posicionados en las articulaciones son reflectores que iluminados por flashes sincronizados de luz infrarroja, son capturados por una o mas cámaras. También pueden ser utilizados marcadores de otros tipos.

El sistema Optotrak, de Northern Digital, utiliza marcadores LED sincronizados que son rastreados por cámaras que son sensibles al efecto infrarrojo. Un software es el encargado de generar las coordenadas 3D. Un problema generado por los marcadores LED es que no se recomienda su utilización por largos periodos de tiempo.

El sistema ELITE, de Bioengineering Tech., utiliza marcadores reflexivos pasivos (hemisferios plásticos cubiertos con material reflexivo) con cámaras electrónicas CCD con LED´S alrededor de los lentes. Un sistema de software utiliza algoritmos de reconocimiento de formas (del tipo Shape from shading) para ayudar al proceso de seguimiento de los sensores.

Una de las ventajas de la utilización de sistemas ópticos es la alta taza de muestreo, que permite capturar movimientos rápidos como en los utilizados en las artes marciales y en los deportes olímpicos. Una taza de muestreo depende básicamente de la calidad de definición de las cámaras utilizadas en el proceso. A mayor resolución que posean las cámaras mayor será la taza de muestreo,

alcanzando unos 200 FPS utilizando cámaras de alta resolución y velocidad de captura.
Otra ventaja de los sistemas ópticos es la libertad que ofrece al actor al ejecutar movimientos. Por lo contrario en los sistemas magnéticos, donde el actor es cubierto de transmisores donde impiden la ejecución de algunos movimientos.

Una gran desventaja de los sistemas ópticos es la cuando los objetos (oclusión) son muy pequeños o cuando existen muchos actores en la escena y se encuentran muy juntos unos de otros. Implicando la utilización de muchas mas cámaras y de reflectores para corregir este problema. Pero al existir un mayor número de cámaras genera un mayor tiempo de procesamiento por parte del computador y aumentar el número de reflectores aumenta la dificultad de identificar los reflectores.

4.3.4. Sistemas Magnéticos. Los sistemas magnéticos de captura se caracterizan por la velocidad de procesamiento de los datos capturados (tiempo real). Este tipo de sistema, posee un conjunto de receptores ubicados en las articulaciones. Estos receptores envían la posición 3D y la orientación de la articulación a través de una antena transmisora que emite una señal de pulso. Cada receptor necesita un cable para conectar la antena.

Algunas ventajas de los sistemas magnéticos es su bajo costo computacional para procesamiento de los datos capturados, mayor precisión de los datos (no existe problema de oclusión). Con una taza de muestreo que oscila entre los 100 FPS, los sistemas magnéticos es uno de los métodos más simples para realizar capturas de movimiento.

La mayor desventaja de este tipo de sistema son los diversos cables que conectan los receptores a la antena. Estas conexiones limitan el movimiento del actor limitando la representación de movimientos complejos y representados con naturalidad.

Otra desventaja es la interferencia producida por metales objetos de metal próximo al área de captura.

4.4. FUENTE EMISOR-RECEPTOR

Una clasificación interesante, propuesta en [Mulder 94], divide los sistemas de captura en tres categorías:

- Inside-in.
- Inside-Out.
- Outside-in.

La semántica de esta clasificación describen en la primera palabra la localización del los sensores (inside, sensores que se encuentra en el cuerpo del actor y outside fuera del actor) la segunda palabra indica la posición de la fuente emisora (por ejemplo, una cámara en los sistemas ópticos), en relación con el cuerpo del actor.

4.4.1. Sistema Inside-in. Este tipo de sistema, los sensores y la fuente transmisora están localizados en el cuerpo del actor. Un ejemplo de este sistema de este tipo es el PowerGlove (Nintento-Mattel), que utiliza sensores flexibles para captar orientaciones. Los sensores de este tipo de sistema poseen generalmente un bajo “Factor forma”, y por consecuencia son mas indicados para capturar movimientos de objetos pequeños (dedos de las manos, gestos faciales etc.). Los datos generados carecen de información 3D, brindado solamente orientaciones relativas.

4.4.2. Sistema Inside-out Magnéticos. Los sistemas donde los sensores están conectados al cuerpo del actor y que responden a las señales emitidas por una fuente emisora externa. Los sistemas magnéticos y acústicos se encuentran en esta categoría. El espacio de trabajo es limitado debido al uso de fuentes

externas. Además el “Factor forma” mas alto restringe su utilización las partes mayores del cuerpo.

4.4.3. Sistema Outside-in. En este tipo de sistema, la fuente emisora esta localizada en el cuerpo del actor, en cuanto los sensores externos capturan las señales. En los sistemas ópticos, la fuente emisora esta recubierta de un material reflexivo, que reflectan la luz emitida por flashes. Estas reflexiones son capturadas por los sensores que son cámaras. Los sistemas de este tipo sufren el problema de la oclusión y a un espacio de trabajo limitado.

4.5. ADQUISICIÓN DIRECTA - INDIRECTA

Esta es una clasificación importante por que ofrece una indicación de cómo son obtenidos los datos capturados, de forma directa o indirecta.

Los sistemas de adquisición directa son aquellos cuyos datos obtenidos no necesitan de un procesamiento posterior a la captura. En general los sistemas de adquisición directa poseen una taza de muestreo muy alta. Están incluidos en esta categoría los sistemas mecánicos, acústicos y magnéticos. A pesar de no necesitar un pos-procesamiento de los datos, los datos capturados generalmente se les debe aplicar un proceso de filtraje para mejorar la calidad y eliminar ruidos.

Los sistemas de adquisición indirecta permiten mayor libertad de movimiento y poseen una alta taza de muestreo. Pero debido a su alta tecnología son extremadamente caros. Este tipo de sistemas son generalmente analizados a través de un robusto sistema de software que hay que agregar al producto final. Estos sistemas son los más recomendados para capturar movimientos complejos y rápidos. Los sistemas ópticos son los integrantes de esta clasificación.

4.6. COMPARACION DE LOS SISTEMAS DE CAPTURA

Tabla **1**. Comparación sistemas de captura

	Sistemas Mecánicos	Sistemas Ópticos	Sistemas Magnéticos
Características principales	Conjunto de potenciómetros	Conjunto de cámaras y de reflectores	Transmisores magnéticos
Principales ventajas	Alta taza de muestreo, campo de trabajo ilimitado	Alta taza de muestreo, libertad de movimiento	Procesamiento en tiempo real, bajo costo de equipamiento
Principales desventajas	Limitación de los movimientos.	Oclusión de los reflectores, alto costo de equipamiento, elevado procesamiento computacional	Restricción de los movimientos, limitado número de marcadores, interferencia externa
Taza de Muestreo	> 120 FPS	> 200 FPS	100 FPS
Ejemplos de Sistemas	Animatton, Digital Monkey, Power Glove	ExpertVision HiRES 3-D System, Multitrax Motion Capture System, OPTOTRAK	A Flock of Birds, ULTRATRAK, FASTRAK, INSIDETRAK
Software Compatibles	Alias | Wavefront, Jack, 3D Studio Max, SoftImage	ZoeTrax, The Creative Motion Editor, PowerAnimator V7	Alias | Wavefront, 4Dvision, SoftImage, Kinemation 3.0, 3D Studio MAX, PowerAnimator V7, Photo4D
Precio	US$ 1,000 a US$ 35,000	US$ 20,000 a US$ 150,000	US$ 5,000 a US$ 70,000

5. ELECCIÓN DEL SISTEMA HARDWARE

5.1. SENSORES

Las características que deben poseer el conjunto de sensores son:

- Portabilidad: El sistema debe permitir el fácil manejo y traslado dentro de un espacio, evitando el exceso de peso y su tamaño.

- Rango de adquisición: Los sensores deben tener la capacidad de adquirir un buen rango del movimiento generado por los puntos de inflexión.

- Inmunidad al ruido: El sistema debe ser altamente inmune a los ruidos generados en el entorno de trabajo, permitiendo la generación de capturas de movimiento continuas y sin sobresaltos

- Costo: Debido a que los sistemas comerciales de captura de movimiento poseen altos costos para su adquisición, este sistema debe cumplir características esenciales para la captura pero evitando que exceso de costo en su desarrollo.

- Sensibilidad: Debido a que los movimientos generados por los puntos de inflexión del brazo poseen altos rangos de medida, no se puede descuidar la capacidad de detectar pequeños movimientos realzados.

- Taza de muestreo: Los sensores deben brindar una alta rata de muestreo, a causa de que los movimientos realizados por el brazo pueden ser efectuados en cortos tiempos.

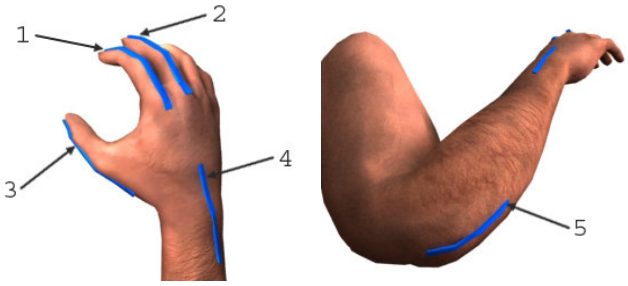
De acuerdo a la tabla Nro. 1 el sistema mas viable frente a las características necesarias para el desarrollo del sistema es la implementación de sensores mecánicos, debido a que permiten altas tazas de muestreo, campo de trabajo ilimitado, y costos bajos respecto a los otros sistemas.

Dentro de los sistemas de captura de movimiento mecánicos los sensores que se implementan en esta tipología son:

- Potenciómetros de precisión.
- Flexómteros.

El guante cuenta con 5 sensores ubicados en puntos críticos para medir la postura de la mano y el brazo. Los sensores al ser sometidos a una deformación cambian su resistencia y a su vez alterando la resistencia que fluye a través de los mismos. Los sensores miden el ángulo de deformación de cada punto de inflexión (Ver Ilustración Nro. 4.1.2). Este sistema no permitirá medir rotaciones de los dedos ni de la muñeca, ni del brazo. Únicamente, realizara medidas representadas en ángulos en un solo eje de rotación.

Ilustración **2**. Ubicación de los sensores en el brazo.



.

Los movimientos a capturar son:

- Dedo índice
- Dedo corazón
- Dedo pulgar
- Muñeca (1 eje)
- Brazo.

Dentro de las tipologías de arreglo de sensores para detectar el movimiento se encontraron los siguientes sistemas:

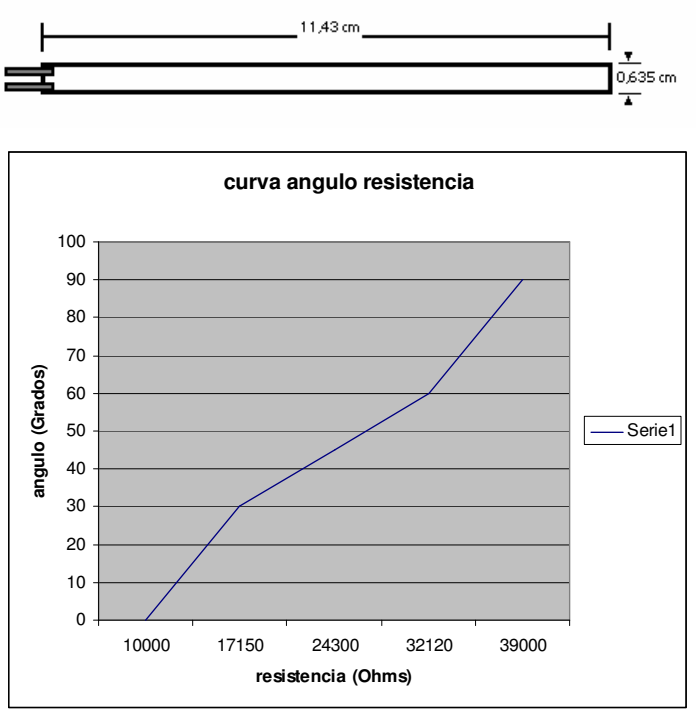
Una de las alternativas para detectar flexiones es por medio de fibra óptica, se coloca un emisor de luz en un extremo de la fibra, el hilo óptico se ubica por toda la trayectoria del dedo desde el falange hasta la falangeta, realizando un doble camino, el diodo al emitir luz a través de una de las puntas de la fibra, genera un haz de luz que recorre todo el hilo óptico, si esta fibra sufre algún deflexión en su forma, la dirección del haz luz al otro extremo de la fibra cambia. Para detectar el ángulo de deflexión se ubica un sensor que genera una variación de voltaje de acuerdo a la inclinación de haz de luz. Este método es muy costoso debido a que los dispositivos de medición no son comerciales.

Otra opción es generada a partir del efecto hall, que su principio básico es que un sensor detecta la variación del flujo magnético convirtiéndolo en variación de voltaje, en cada articulación se ubica un elemento generador de campo magnético al extremo final del falange y el sensor del efecto hall en la parte inicial de falangina, al moverse la falangina respecto del falange provoca que el campo magnético varíe, generando el sensor una señal de voltaje respecto de la lejanía o cercanía del generador de campo magnético.

La solución menos probable es generar un mecanismo que conecte cada articulación con una serie de potenciómetros lineales que variarán su resistencia respecto al ángulo de flexión. Esta opción se descarta debido a la limitación de movimientos.

La opción que más razonable es la utilización de un similar a las galgas extensiométricas, este sensor se comporta como una galga extensiométrico a gran escala midiendo unos 11,43cm, respecto de los 4cm de la galga, este sensor es llamado FLEXSENSOR, y fue desarrollado a inicios de 1990 por la compañía Mattel, que creó un sistema de captura denominado POWER GLOVE para la compañía Nintendo.
El comportamiento y sus características están descritos en las siguientes gráficas:

Ilustración **3**. Dimensiones y comportamiento de la resistencia del sensor

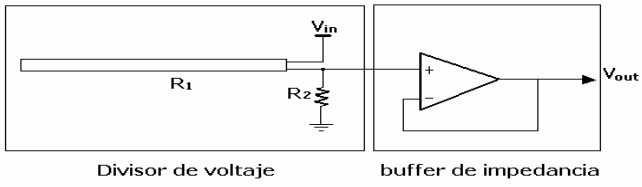


El voltaje de salida está descrito por la siguiente ecuación:

$$v_{out} = v_{in}\left(\frac{R_2}{R_1 + R_2}\right)$$

Donde:

Ilustración **4**. Diagrama electrónico para el acondicionamiento de la señal generada por el sensor



5.2. SISTEMA DE ADQUISICIÓN DE DATOS

La arquitectura de un sistema de adquisición de datos es muy simple y consta de 3 etapas:

Ilustración **5**. Diagrama en bloques de un sistema de adquisición de datos

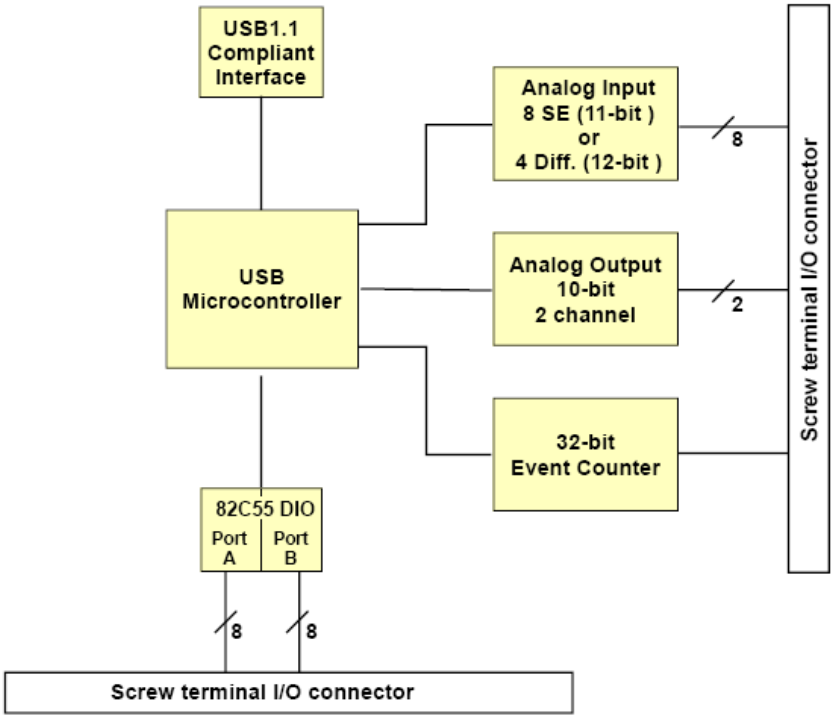
Existen muchos sistemas de adquisición de datos, las características necesarias para el sistema son:

- Resolución de captura: es la capacidad de poder convertir un dato análogo de voltaje en un valor digital equivalente.

- Velocidad de lectura por canal: es la velocidad necesaria para que el multiplexor o intercambiador de canales de entrada recorra todas las entradas de voltaje análogo.

El sistema de adquisición de datos seleccionado para realizar el proyecto es USB-1208LS creado por Measurement Computing que genera un sistema análogo y digital con una interfase de comunicación con el puerto USB. Posee 8 entradas analógicas de una sola polaridad con 12-bit de resolución de conversión. Este sistema posee una taza de muestreo superior a 1200 muestras por segundo. Posee compatibilidad con cualquier lenguaje de programación.

Ilustración **6**. Diagrama interno del sistema de adquisición de datos



6. INTRODUCCIÓN A LAS LIBRERÍAS GRÁFICAS

El desarrollo de los sistemas gráficos 3D que actualmente son las herramientas para generar aplicaciones tridimensionales surge del gran avance tecnológico en los computadores.

Desde la creación del computador muchas personas se interesaron por generar ambientes gráficos asistidos por el computador. Una de estas ramas interesadas en que los gráficos 3D surgieran como una alternativa generadora de contenido visual fueron los grandes consorcios cinematográficos y gracias a ello se crearon las bases fundamentales para el desarrollo de librerías gráficas.

Hacia 1961 un estudiante del MIT (Massachussets Institute of Technology), Ivan Sutherland creó un programa de computador llamado Sketchpad. Mediante un lápiz óptico, Sketchpad permitía dibujar formas simples en la pantalla del computador, guardarlas y luego visualizadas. Todo era basado en un programa de vectores. Hacia la misma fecha otro estudiante del MIT, Steve Rusell, creó el primer video juego llamado SpaceWar.

E. E. Zajac, un científico de Bell Telephone Laboratory (BTL), creo una película 3D llamada “Simulation of a two-giro gravity attitude control system” en 1963. En el computador donde se generó la película, Zajac mostró como el comportamiento de un satélite puede alterar la orbita de la tierra. Esta animación se creó en una computadora IBM 7090.

Hacia mediados de los 60’ el interés de los gráficos por computador fue creciendo en forma significativa, corporaciones como TRW, Lockheed-Georgia, General Electric y Sperry Rand iniciaron desarrollos en el campo de los gráficos por

computador. IBM respondió a esta demanda creando la estación gráfica IBM 2250 que fue el primer computador gráfico comercial.

En 1966 Sutherland del MIT invento el primer computador controlado por un HMD (Head Mounted Display). Llamado “La espada de Damocles”, este mostraba dos imágenes separadas, una para cada ojo. Esto permitió ver escenas estereoscópicas en 3D por computador, las imágenes eran una estructura de alambre (wireframe) de los objetos presentes en la escena.

El primer mayor avance en gráficos 3d por computador hacia finales de los 60’, fue creado en la Universidad de Utah, el algoritmo para ocultar geometrías (hidden-surface). Para representar un objeto 3D en la pantalla, el computador debía determinar que caras debían se ocultadas del objeto de acuerdo a la perspectiva del observador y cuales eran visibles cuando el computador creaba la imagen (render).

Para lograr mayor realismo en las imágenes 3D, que para esa época era un conjunto de bordes filosos que conformaban un objeto, Henri Gouraud en 1971 presentó un método para crear la apariencia de una superficie curva interpolando el color a través de los polígonos. Este método de sombreado es conocido como el método de sombreado Gouraud.

Ed Catmull recibe su Ph. D. en ciencias de la computación en 1974 y su tesis se abarco el mapeo de texturas (Texture Mapping), Z-Buffer y el dibujado de superficies curvas. El mapeo de texturas consiste en obtener una imagen plana 2D y aplicada aun objeto 3D generado por computador. El z-buffer es un área de memoria dedicada a almacenar la profundidad de cada píxel presente en la escena y elimina los que no son visibles.

En el ’74 Phong Bui-Toung, un programador de la Universidad de Utah. Desarrollo un nuevo método de sombreado que corrige algunos problemas generados en el algoritmo de sombreado de Gouraud, denominado algoritmo de sombreado de Phong. Su método se basa en obtener exactamente los brillos y las sombras generadas por el objeto 3D. El inconveniente con este método es que puede ser 100 veces más lento que el método de sombreado Gouraud.

En el ’76 James Blinn desarrolla una nueva técnica similar al mapeo de texturas. Pero, en lugar de un simple mapeo de la imagen 2D en el objeto 3D, los colores de la imagen son usados para hacer que la superficie contenga rugosidad. Para hacer esto, se utiliza una imagen monocromática, donde existen áreas blancas se genera una saliente y las áreas negras generan un hundido. Este método se denomina Bump Mapping.

A partir del ’77 películas como Star Wars Episodio IV (1977), Alien (1979), The Black Hole (1979), Tron (1982). Involucraron gráficos 3D generados por computador, segmentos cortos pero que en su momento revolucionaron el mundo de las artes cinematográficas.

En enero de 1984, Apple Computer presenta el primer computador Macinthosh, que fue el primer computador personal en usar interfase gráfica.

Antes de 1990 los gráficos 3D se convirtieron en una tecnología crítica para varios mercados. Las necesidades de múltiples casas desarrolladoras de software 3D condujeron a desarrollar una API de gráficos 3D común. Esta nueva API, fue llamada OpenGl (Open Graphics Library). Descendiente directo de SGI (Silicon Graphics Inc.) IRIS GL. OpenGl surge como una librería de recursos gráficos 3D de lenguaje por procedimientos. OpenGl debía ser una API avanzada que podría

ser ejecutada eficientemente en múltiples computadoras. Y su estructura sería controlada por un comité conocido como ARB (Architecture Review Board). Los integrantes de este comité fueron Digital Equipment Corporation, Intel, Microsoft y Silicon Graphics.

En mayo de 1990, Microsoft lanza al mercado Windows 3.0. Compuesto de una GUI (Graphical User Interface) estructuralmente similar a la de Apple Macintosh.

Con la creación de Windows 3.1 Microsoft intento remediar defectos presentes en la versión 3.0, pero esto no fue suficiente para que OpenGL corriera sobre el sistema operativo de Microsoft. Para solucionar Microsoft remedió estos problemas en Windows NT, permitiendo la funcionalidad de esta librería en ambiente Windows.

Para apoyar el desarrollo de videojuegos en computadoras que funcionaban con sistema operativo Windows 95, Microsoft entre 1995 y 1996 decide no utilizar la tecnología de OpenGl proporcionada en Windows NT, y en respuesta compra a Rendermorphics Ltda., adquiriendo la API 3D conocida como RealityLab. Microsoft rediseño los drivers y anuncio la nueva API gráfica de modo inmediato Direct3D.

A partir de este evento significativo inicia la competencia entre API´s gráficas 3D.

6.1. LIBRERÍAS GRÁFICAS 3D

Las API´s gráficas más utilizadas actualmente son:

- OpenGl.
- Direct3D.
- Java3D.

Es claro que existen muchas librerías gráficas 3D que con el tiempo fueron desechadas o reemplazadas por tecnología más versátiles, como el proyecto llamado Fahrenheit que pretendía unir Opengl con Direct3D, o Glide que fue la primera API gráfica escrita exclusivamente para las primeras tarjetas de video.

En casos como Java3D que es una API que no puede realizar aceleración por hardware por si sola, se apoya en utilizar OpenGl o Direct3D que son los que realmente pueden explotar todos los recursos brindados por la tarjeta de video, por lo tanto no será caso de discusión en este documento.

6.2. OPENGL

"OpenGL ("Open Graphics Library") es una interfase de software para gráficos generados por Hardware. La interfase consiste en un grupo de cientos de procedimientos y funciones que permiten al programador especificar los objetos y operaciones envueltos en la producción de imágenes de alta calidad gráfica, específicamente imágenes de tres dimensiones" [glspec20].
Es una API definida para multilenguaje y multiplataforma, para generar aplicaciones gráficas 3D y 2D. Esta interfase esta compuesta de 250 diferentes llamados a funciones y cada una puede ser usada para dibujar complejas escenas tridimensionales a partir de simples primitivas. Es muy popular en la industria de los videojuegos y compite con su archirival Direct3D. OpenGl es utilizado ampliamente en CAD, realidad virtual, visualización científica, etc.

OpenGl o "GL" se ocupa de generar una imagen dentro de un búfer ubicado en el chip de video que es el encargado de mostrar la imagen en pantalla. Esto no es soportado por otros periféricos que algunas veces son asociados con gráficos por hardware. Como el Mouse y el teclado.

GL dibuja primitivas sujeto a un número determinado de modos. Cada primitiva es un punto, un segmento de línea, polígono, o un rectángulo-píxel. Cada modo puede ser manipulado independientemente y los parámetros de uno no pueden afectar a los demás.

Las primitivas pueden ser definidas en un grupo de uno o más vértices. Un vértice define un punto, y dos pueden definir un borde (Edge), o una esquina de un polígono. Los datos (coordenadas de posición, color, normales, y coordenadas de

textura) son asociados con un vértice y cada vértice es procesado independientemente, en un orden específico y siempre en el mismo sentido.

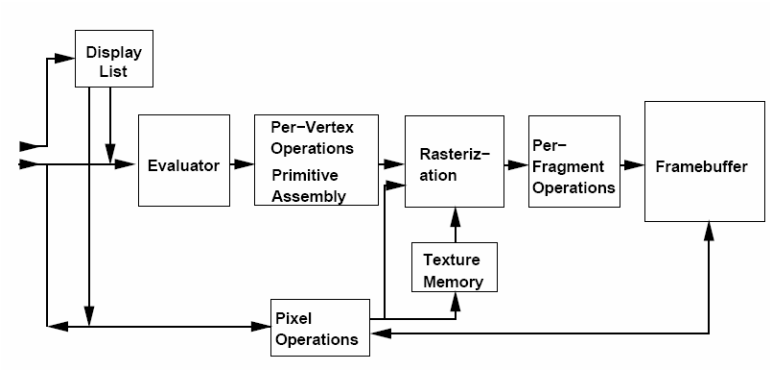
GL provee un control sobre operaciones fundamentales de gráficos 2D y 3D. Esto incluye la especificación de parámetros como matrices de transformación, coeficientes de ecuaciones para iluminación, métodos de suavizado de imagen, etc. Pero no incluye funciones para describir una geometría compleja.

El modelo para la implementación de comandos GL es cliente-servidor. Esto es, un programa (el Cliente) usa los comandos, y estos comandos son interpretados y procesados por GL (el Servidor). El servidor quizás pueda o no operar en el mismo computador del cliente. En pocas palabras, GL es una red invisible.

Los comandos efectuados en la aplicación principal (framebuffer) son finalmente controlados por el Sistema de Ventanas que asignan un espacio de memoria para los recursos utilizados por el framebuffer. El Sistema de Ventanas determina que partes del framebuffer pueden acceder GL de forma inmediata.

6.2.1. OpenGl Pipeline. La ilustración Nro. 7 muestra un diagrama esquemático de GL. Los comandos ingresan al GL por la izquierda. Algunos comandos especifican objetos geométricos que serán dibujados mientras que otros controlan cómo los objetos son manipulados en varias etapas. La mayoría de comandos se puede acumular en una lista para ser procesados por GL mas tarde. De lo contrario los comandos se envían eficientemente por el bus de procesos (desde ahora Pipeline).

Ilustración **7**. Diagrama interno del Pipeline de OpenGl



El primer paso provee un método eficiente para aproximar curvas y superficies geométricas evaluando funciones polinomiales a partir de los datos de entrada. El siguiente paso opera en las primitivas geométricas descritas por vértices, puntos, segmentos de líneas y polinomios. En este paso los vértices son transformados e iluminados. Las primitivas son recortadas de acuerdo la ubicación de la vista para prepararlos para el próximo paso, es la generación de una imagen de mapa de bits (rasterization). El rasterizer produce una serie de direcciones al framebuffer y valores usando una descripción 2D de un punto, un segmento de línea, o un polígono. Cada fragmento producido es llevado a la próxima etapa que ejecuta operaciones con los fragmentos individualmente antes que alteren el framebuffer. Esas operaciones incluyen actualizaciones condicionales al framebuffer basado en los valores de profundidad actuales y los nuevos, para agregar el efecto de profundidad, mezcla con los fragmentos de colores almacenados al igual que el enmascarado y otras operaciones lógicas.

Existe una forma de evitar el procesamiento de vértices enviando bloques de fragmentos directamente al bloque que procesa fragmentos individuales. Que

eventualmente generan un bloque de píxeles que pueden ser escritos en el framebuffer, pueden ser leídos y copiados a cualquier dirección de memoria.

OpenGl fue diseñado solo para generar salidas. Está provisto únicamente de funciones de dibujado. Dentro de esta API no esta concebido el concepto de sistema de ventanas, audio, impresión, periféricos (teclado, Mouse, etc.).

La construcción y desarrollo de OpenGl esta regida por la ARB (Architecture Review Board) que es compuesta de un gran número de compañías que determinan de acuerdo a sus intereses las especificaciones de la librería. Esto implica que para producir una nueva versión se tomen grandes periodos de tiempo entorpeciendo el proceso de desarrollo y evolución de GL.

Una de las últimas ideas novedosas que se agregó a la nueva versión 2.0 del GL fue propuesta por 3Dlabs que realizó adiciones significativas al estándar, la más significativa son los GLSL (OpenGL Shading Language). Esto permitirá al programador reemplazar el llamado a la función que procesa vértices y fragmentos utilizando shaders escritos en lenguaje c, con estos shaders se pueden generar efectos como bump-mapping, distorsión por combustión, olas de agua y ondas. Todo esto realizado directamente por el procesador de gráficos, logrando efectos con mucha rapidez.

6.3. DIRECT3D

Direct3D es una librería grafica que permite a los desarrolladores de aplicaciones acceder a todos los recursos disponibles en la tarjeta de video.

En Direct3D el programador tiene dos opciones: desarrollar la aplicación con un pipeline fijo o uno programable.

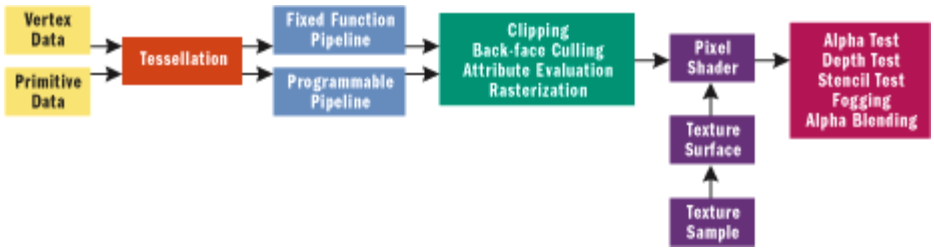
El fijo depende de algoritmos estandarizados por Direct3D. Estas funciones fijas son expuestas a través de un conjunto fijo de valores enumerados similares a OpenGl (Iluminación, mezcla de transparencias, etc.). Algunas funciones pueden ser aceleradas por el hardware gráfico, dependiendo de las prestaciones que brinde la tarjeta. Cuando se usa el pipeline fijo el programador primero debe chequear si la tarjeta gráfica soporta una característica en particular, debido a que algunas tarjetas no soportan rutinas descritas en la librería de Direct3D.

En el pipeline, en lugar de seleccionar una enumeración predefinida de funciones y solicitar a Direct3D que ejecute una serie de funciones. El programador puede definir sus propios algoritmos. La aplicación en tiempo de ejecución compilará dinámicamente el algoritmo para el hardware. Para esta funcionalidad Direct3D tiene un compilador denominado JIT (just-in-time), el cual hace parte explicita del controlador del dispositivo hardware. Direct3D funciona como una maquina virtual de gráficos, el cual, eficazmente virtualiza el procesador de gráficos (GPU) con un grupo de instrucciones gráficas personalizadas.

6.3.1. Direct3D Pipeline. El pipeline de Direct3D es similar a la arquitectura del hardware del computador, Direct3D utiliza la arquitectura utilizada en los microprocesadores donde el microprocesador comienza ejecutando una segunda

instrucción antes que la primera haya sido finalizada (pipelining) y la otra es ejecutar las acciones en paralelo (parallelizing). Los algoritmos expuestos a través de Direct3D son lógicamente organizados dentro del pipeline.

Ilustración **8**. Diagrama interno del Pipeline de Direct3D



El pipeline debe ser visto como un grupo de algoritmos que opera con cantidades geométricas 3D, el cual en el caso de Direct3D son predefinidas como vértices y primitivas. El propósito principal del pipeline es convertir datos geométricos en una imagen, que es dibujada por la tarjeta de video en la pantalla.

La etapa de tesellation es utilizado para convertir en triángulos (tessellate) un número fijo de geometrías de alto orden predefinidas por Direct3D, que incluyen patches triangulares, patches rectangulares y patches multidimensionales. Actualmente la tesellation no es programable, de esta manera Direct3D no suministra un mecanismo para generar geometría procedimental en la tarjeta de video. La geometría procedimental provee beneficios con respecto a minimizar la cantidad de datos enviados a través del bus de datos. En un futuro muy cercano ya será posible que la tesellation sea programable, actualmente en la consola de video juegos Xbox360 el fabricante de tarjetas de video ATI implemento esta funcionalidad, se espera que sea implementado para PC.

La etapa de transformación e iluminación en el pipeline transforma las posiciones de los vértices y normales del sistema de coordenadas del modelo a las coordenadas del mundo y luego de la cámara. Esto ocurre a través de las transformaciones de mundo y de vista. Los cálculos de iluminación por vértice son ejecutados para determinar los brillos y las componentes de color. Las posiciones de los vértices son entonces transformadas por la matriz de proyección para generar, ya sea una visualización en perspectiva, ortogonal y otro tipo de proyección.

En la etapa de procesado de vértices, se tiene la opción usando una función fija para la asignación de texturas o una programable mediante los píxel shaders que determinan el color de cada píxel. Con la función fija el multi-texturing es procesado a través de una serie de pasos para establecer el color y el valor de transparencia. Por otro lado los píxel shaders proveen mucho mas control logrando mediante pequeñas rutinas de código asignar colores a los píxeles y valores de transparencias. Además, estos pequeños algoritmos procesan efectos como Bump-Mapping, sombreado, mapeo ambiental.

El procesado de frame buffer involucra un conjunto de regiones de memoria conocidas como dibujado de superficies, buffer de profundidad y el esténcil buffer. Durante este paso, una serie de cálculos son realizados para determinar los valores de la profundidad, la transparencia y el esténcil. El buffer de profundidad es otra optimización de dibujado para remover líneas y superficies escondidas o no visibles dentro del campo de visión. La profundidad se determina para considerar los píxeles no visibles y que no sean dibujados y esta operación calcula utilizando el z-buffer o el w-buffer.

Direct3D viene de una familia de librerías denominada DirectX. DirectX proporciona un enlace directo con el hardware en el sistema de Microsoft Windows. Cada API controla un listado de funciones de bajo nivel que accedan al hardware presente y si este no existe es emulado por software. Estas funciones incluyen soporte para aceleración de gráficos 2D y 3D, control de muchos dispositivos de entrada, funciones para mezclar y filtrar sonido, controlar redes y videojuegos de múltiples jugadores, y proveer control sobre formatos de compresión para multimedia. Las componentes que hacen parte de la librería son:

- DirectDraw.
- Direct3D.
- DirectInput.
- DirectSound.
- DirectMusic.
- DrectPlay.
- DirectShow.

DirectX utiliza dos tipos de controladores, la capa de abstracción del hardware (HAL) y la capa de emulación del hardware (HEL), para enviar solicitudes al dispositivo hardware. Cuando DirectX es inicializado, chequea si el hardware puede existe y si soporta ciertas capacidades. Si el hardware soporta las características necesarias, el HAL será el encargado de acceder al hardware, de lo contrario es utilizado el HEL para emular las capacidades a través de software.

6.4. OPENGL VS. DIRECT3D

6.4.1. Facilidad de Uso. A comienzos del Direct3D fue conocido como herramienta torpe para el manejo de gráficos 3D, debido a que algunas operaciones requerían muchas configuraciones para poder ser visualizadas, a diferencia de OpenGl que mediante instrucciones simples lograba obtener mejores resultados.

A medida que Direct3D evolucionó a la versión 8.0, muchos cambios fueron realizados, y todo el desarrollo de esta API fue basada en el COM de Microsoft Windows. Donde la base de las funciones estaba orientada a objetos (una textura es un objeto, por ejemplo). OpenGl por lo contrario posee una maquina de estados que son accedidas a través de funciones escritas en C.

Direct3D esta más asociado al hardware que OpenGl. Por ejemplo, si se minimiza la aplicación y se restaura de nuevo se genera un efecto de pérdida Dispositivo (LostDevice), implicando que el programa debe recargar todas las texturas y tudas las constantes. OpenGl hace eso transparente para el programador, realizando este evento automáticamente provocando menos control de la aplicación.

6.4.2. Portabilidad. Direct3D es una API atada a una plataforma específica: Microsoft Windows. La última implementación realizada de esta API fue a la consola de juegos propietaria de Microsoft XBox. Direct3D está tan altamente ligado a esta plataforma que pensar implementarlo en otros sistemas operativos requeriría muy arduo trabajo y en la práctica es imposible y de poco interés para Microsoft. OpenGl, por otra parte, esta implementado en una amplia gama de plataformas incluyendo Windows, sistemas basado en UNIX, Linux, Macintosh, Nintendo y GameCube. En general, casi todos los sistemas operativos modernos

capaces de producir gráficos 3D tienen un controlador de OpenGl. Esto hace que Direct3D frente a OpenGl tenga una gran desventaja. Aunque la elección de la API depende de los intereses de quien lo programe. En cuanto a video juegos el objetivo del mercado es determinar el sistema operativo mas utilizado y Microsoft Windows es la plataforma mas difundida en el mundo.

6.4.3. Extensiones. El mecanismo de extensión es quizás uno de los puntos a favor de OpenGl debido a que cualquier fabricante de tarjetas de video puede implementar funciones específicas que acceden al hardware creado por ellos, permitiendo incluir nuevas funcionalidades a los gráficos 3D de forma rápida. Pero también esto conlleva a la confusión debido a que existen diversos fabricantes de hardware llevando a la generación de múltiples extensiones. Muchas de estas extensiones son periódicamente estandarizadas por la ARM. Por otra parte, Direct3D es totalmente diseñado y construido por un solo diseñador (Microsoft), que conduce a generar una API más constante. En resumen al generar código con Direct3D, su funcionalidad se puede apoyar en distintas tarjetas de gráficos y el programador no se tiene que preocupar por implementarlo. Mientras que en OpenGl el programador tendría que escribir sistemas individuales implementados para cada tarjeta de video.

6.4.4. Tabla de comparación. A continuación la tabla describe la comparación entre las funcionalidades incorporadas dentro de OpenGl y Direct3D:

Tabla **2**. Tabla de comparación de OpenGl y Directx.

Feature	OpenGL 1.2 Core	Direct3D 7	Direct3D 8
System Mechanics			
Operating System Support	Windows (9x, NT, 2000), MacOS, BeOS, *nix, others	Windows (9x, 2000, CE)	Windows (9x, 2000)
API Definition Control	OpenGL ARB	Microsoft	Microsoft
API Specification	OpenGL Specification	SDK/DDK Documentation and DDK Reference	SDK Documentation
API Mechanism	includes and libraries	COM	COM
Software Emulation of Unaccelerated Features	Yes	No	No
Extension Mechanism	Yes	No	Yes
Source Implementation Available	Yes	Yes	No
Modeling			
Fixed-Function Vertex Blending	No	Yes	Yes
Programmable Vertex Blending	No	No	Yes
Parametric Curve Primitives	Yes	No	Yes
Parametric Surface Primitives	Yes	No	Yes
Hierarchical Display Lists	Yes	No	No
Rendering			
Two-sided Lighting	Yes	No	No
Point Size Rendering Attributes	Yes	No	Yes
Line Width Rendering Attributes	Yes	No	No
Programmable Pixel Shading	No	No	Yes
Triadic Texture Blending Operations	No	No	Yes

Cube Environment Mapping	No	Yes	Yes
Volume Textures	Yes	No	Yes
Multitexture Cascade	No	Yes	Yes
Texture Temporary Result Register	No	No	Yes
Mirror Texture Addressing	No	Yes	Yes
Texture "Wrapping"	No	Yes	Yes
Range-Based Fog	No	Yes	Yes
Bump Mapping	No	Yes	Yes
Modulate 2X Texture Blend	No	Yes	Yes
Modulate 4X Texture Blend	No	Yes	Yes
Add Signed Texture Blend	No	Yes	Yes
Frame Buffer			
Hardware Independent Z Buffer Access	Yes	No	No
Full-Screen Antialiasing	Yes	Yes	Yes
Motion Blur	Yes	No	Yes
Depth of Field	Yes	No	Yes
Accumulation Buffers	Yes	No	No
Miscellaneous			
Picking Support	Yes	No	No
Multiple Monitor Support	No	Yes	Yes
Stereo Rendering	Yes	Yes	No

Es importante aclarar que la tabla anterior fue realizada hacia el año 2000 donde la versión de OpenGL fue 1.2 y solo se evalúa la base de GL. No se tienen en cuenta las extensiones y la de Direct3D la 7.0 y la 8.0.

Actualmente la versión de OpenGl es 2.0, y la de Direct3D es 9.0c.

6.4.5. Implementaciones. Tanto OpenGl como Direct3D son librerías gráficas que se pueden acceder desde múltiples ambientes de programación. OpenGl tiene una gran ventaja al poder ser implementado en distintas plataformas, permitiendo el desarrollo de aplicaciones generadoras de gráficos tridimensionales en muchos campos. Por lo contrario Direct3D solo puede ser accedido en un ambiente de programación bajo Microsoft Windows limitando la diversidad de soluciones.

A causa de que las librerías gráficas poseen un nivel mas bajo de programación que cualquier aplicación, la necesidad de su implementación depende si la aplicación requiere de aceleración de gráficos 3D. Los campos a implementar más comunes son:

- Aplicaciones CAD-CAM.
- Aplicaciones de Modelado 3D.
- Simuladores de Entornos Virtuales.
- Motores de Juegos.

7. DESARROLLO DEL SISTEMA SOFTWARE (REQUERIMIENTOS)

7.1. OBJETIVO GENERAL

Desarrollar una aplicación software capaz de recibir, convertir, representar, manipular y almacenar los movimientos detectados por un sistema Hardware de adquisición de datos, agregando valor al proceso de visualización e interpretación de los datos a través del uso de gráficos tridimensionales.

7.2. OBJETIVOS ESPECIFICOS

- Utilizar librerías gráficas para el desarrollo de un visualizador gráfico tridimensional.

- Dotar al visualizador gráfico de herramientas que faciliten la visualización de los objetos contenidos en el espacio a través de funciones como alejamiento o acercamiento, desplazamiento de la pantalla, y rotación del entorno (Zoom, Pan, TrackBall).

- Incluir un brazo tridimensional que responda a las señales generadas por el sistema Hardware encargada de la captura de movimiento.

- Permitir el almacenamiento en disco de datos capturados por el sistema hardware.

- Permitir la reproducción de datos almacenados en disco de la captura y su visualización a través del brazo tridimensional.

7.3. ALCANCE

La aplicación a desarrollar estará limitada en los siguientes aspectos:

- El visualizador gráfico solamente se podrá utilizar con el sistema Hardware de captura diseñado para los fines del proyecto, no se podrán adaptar otros sistemas Hardware de captura.

- El objeto tridimensional que responderá a los datos generados por el sistema Hardware será únicamente un brazo tridimensional. No se podrá adaptar otro tipo de objeto 3D que responda a estos eventos.

- Previo a la visualización de los elementos tridimensionales y la captura de movimiento, se deberá calibrar la parte Hardware, de otra forma, no se garantiza óptimo desempeño.

7.4. DEFINICIÓN DEL SISTEMA

El sistema se puede dividir en dos importantes subsistemas:

- Calibración: Se encargará de calibrar y configurar todo el sistema Hardware, permitiendo la posibilidad de observar las propiedades de cada sensor y su estado. Observar características de la etapa de adquisición tales como estado de la tarjeta DAQ (Data Adquisition) y calibración de la misma. Establecerá los valores máximos y mínimos emitidos por los sensores para asociarlos con el movimiento del brazo tridimensional. También estará encargada de la manipulación de la información proveniente de la etapa hardware, filtrado y conversión de datos para su posterior uso en la etapa de visualización.

- Visualización: Permitirá ver en la pantalla del computador la representación de los movimientos capturados por el sistema Hardware en tres dimensiones. Esta aplicación contendrá herramientas básicas para visualización interactiva: Zoom, Pan, TrackBall.

La aplicación debe estar provista de un brazo tridimensional que reaccionará a los movimientos efectuados por el usuario a medida que se capturan datos a través del sistema Hardware, o cuando se reproduzcan datos de captura almacenados en disco.

Finalmente, el subsistema de visualización, tendrá la capacidad de almacenar los movimientos del brazo y guardar los cambios de cada sensor en una estructura de datos que permitirá su posterior visualización y análisis en el sistema de visualización 3D.

7.5. ESPECIFICACIONES SUPLEMENTARIAS (NO FUNCIONALES)

Los atributos de esta aplicación estarán enfocados hacia la funcionalidad, confiabilidad y sobre todo su usabilidad.

Confiabilidad.

La aplicación debe ser lo suficientemente confiable como para asegurar que todos los datos capturados por el sistema Hardware sean totalmente representados por el brazo 3D, además los tiempos de lectura de datos deben garantizar que tanto la captura como la representación sean en tiempo real impidiendo que la información capturada se pierda.

Usable.

Esta es quizá una de las características mas importantes y sobresalientes del sistema, debe garantizar su fácil uso, operabilidad, destacando el aspecto de configuración y ajuste del sistema Hardware para su correcto uso. Es necesaria una interfaz gráfica muy amigable al usuario.

7.6. DEFINICIÓN DE ACTOR

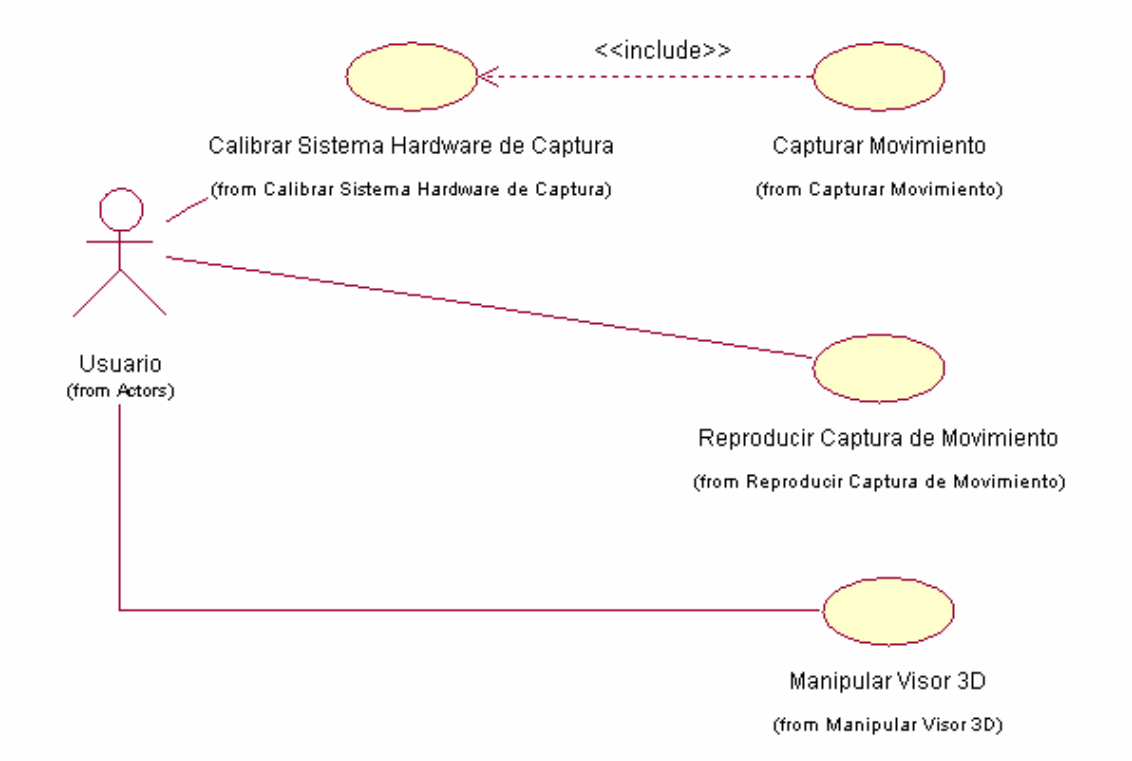
Ilustración **9**. Usuario de Sistema

Usuario del Sistema.

Aquella persona que puede acceder al sistema para su uso. No requiere de conocimientos especiales a nivel de software y/o programación. No tendrá posibilidad de cambiar el código fuente de la aplicación, es un usuario convencional de herramienta de autor.

7.7. DIAGRAMA DE CASOS DE USO

Ilustración **10**. Diagrama de Casos de Uso



En el capitulo Nro. 9 (Anexos) en el punto 1 se describe cada caso de uso y las interfaces que integrarán la aplicación.

8. DESARROLLO DEL SISTEMA SOFTWARE (ANÁLISIS)

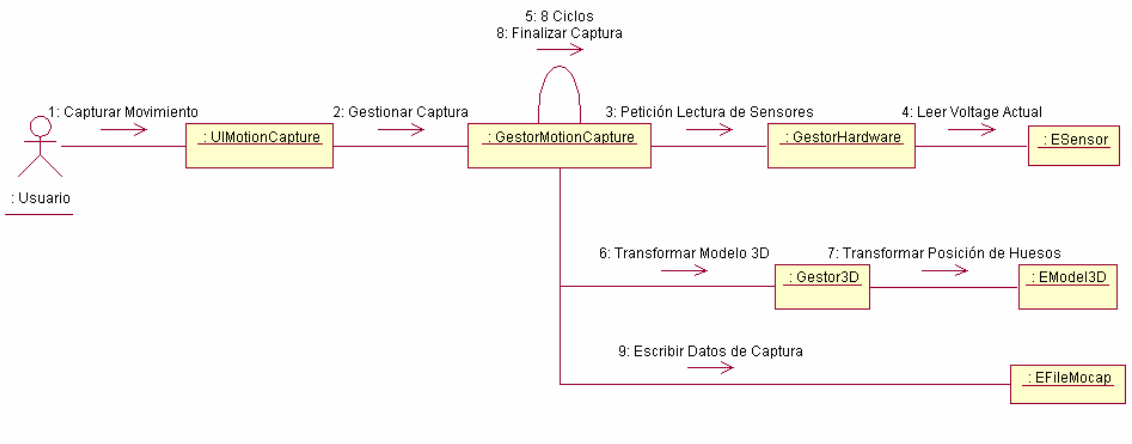
Una Realización de caso de uso – análisis es una colaboración dentro del Modelo de Análisis que describe cómo se lleva a cabo y se ejecuta un caso de uso determinado en términos de las clases de análisis. Por lo tanto hay una realización de caso de uso – análisis para cada caso de uso expresado en el capitulo 9 (Anexos) punto número 1.

En este capitulo de realiza el modelo de análisis al caso de uso número 2 por su complejidad e importancia para el desarrollo del sistema. En el capitulo 9 (Anexos) en el punto 2.

Número: 002

Nombre de Caso de Uso-Análisis: "Generar Captura de Datos"

Ilustración **11**. Caso de Uso-Análisis 02 "Generar Captura de datos"



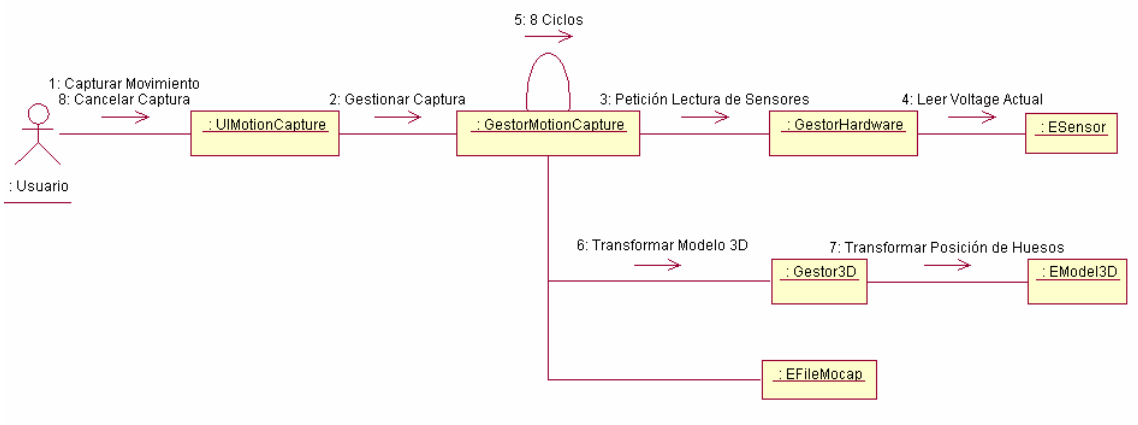
Definición

El Usuario inicia el caso de uso interactuando con la interfaz de usuario, seleccionando la opción de Captura de Movimiento. Esta interfaz de usuario se comunicará con la clase control proporcionándole la orden pulsada, la clase control solicita mediante la interfaz de usuario determinar el fichero donde se guardará el archivo de captura y los parámetros de captura del sistema Hardware. Se inicia el proceso de captura hasta el tiempo determinado por el usuario. Al finalizar la captura de datos la clase control guarda el archivo con los datos capturados y mediante la interfaz de usuario se pregunta si se reproduce la animación de la captura realizada.

Flujo alternativo

El Usuario cancela el proceso de Captura de Movimiento.

Ilustración **12**. Caso de Uso-Análisis 02 “Generar Captura de datos” Flujo Alternativo

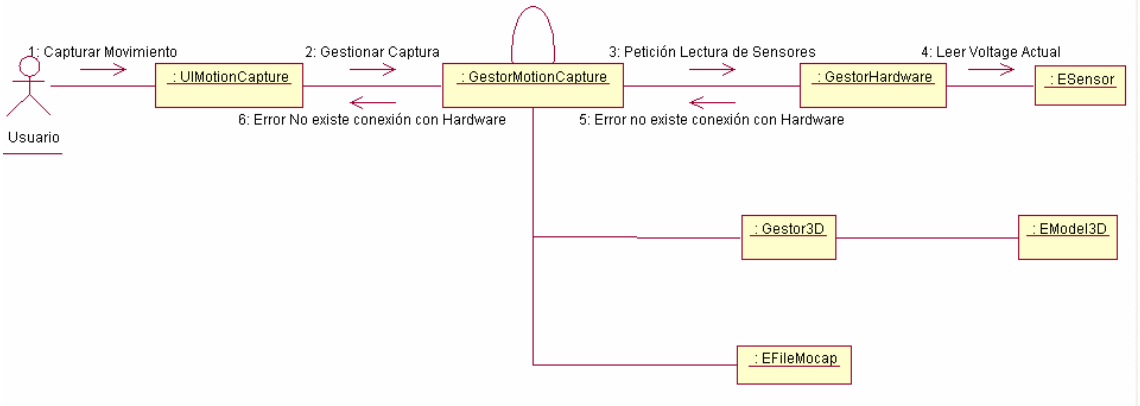


El Usuario inicia el caso de uso interactuando con la interfaz de usuario, seleccionando la opción de Captura de Movimiento. Esta interfaz de usuario se comunicará con la clase control proporcionándole la orden pulsada, la clase control solicita mediante la interfaz de usuario determinar el fichero donde se guardará el archivo de captura y los parámetros de captura del sistema Hardware. Se inicia el proceso de captura hasta el tiempo determinado por el usuario. El usuario mediante la interfaz de usuario decide cancelar el proceso de Captura de Movimiento, la clase control finaliza el ciclo de captura y no almacena los datos capturados en el archivo.

Flujo alternativo

La aplicación pierde conexión con el Sistema Hardware.

Ilustración **13**. Caso de Uso-Análisis 02 “Generar Captura de datos” Flujo Alternativo



El Usuario inicia el caso de uso interactuando con la interfaz de usuario, seleccionando la opción de Captura de Movimiento. Esta interfaz de usuario se comunicará con la clase control proporcionándole la orden pulsada, la clase control solicita mediante la interfaz de usuario determinar el fichero donde se guardará el archivo de captura y los parámetros de captura del sistema Hardware. Se inicia el proceso de captura hasta el tiempo determinado por el usuario. Durante el proceso de Captura la clase control Hardware detecta una pérdida de comunicación con el Sistema Hardware emitiendo un error a la clase control de captura. La clase control de captura determina no continuar con el proceso de captura y comunica al usuario mediante la interfaz de usuario la falla de conexión con el Sistema Hardware ciclo de captura y no almacena los datos capturados en el archivo.

9. DESARROLLO DEL SISTEMA SOFTWARE (DISEÑO)

9.1. REALIZACIÓN DE CASOS DE USO DISEÑO

En esta parte vemos la colaboración del modelo de diseño que describe como se realiza y ejecuta un caso de uso en términos de clases de diseño y sus objetos, conteniendo:

- Diagrama de Clases de diseño, que participan para cada caso de uso.

- Diagramas de interacción, que muestran las interacciones entre los objetos de diseño al realizar el caso de uso, se inician con el envío de un mensaje por parte de un actor externo.

- Descripción textual del flujo de eventos, para clarificar los diagramas de interacción.

En este capitulo de realiza el modelo de diseño al caso de uso número 2 por su complejidad e importancia para el desarrollo del sistema. En el capitulo 9 (Anexos) en el punto 3.

Número: 002

Nombre de Caso de Uso-Diseño: “Generar Captura de Datos”.

Estado: (Liberad/ En Revisión/ En Desarrollo)

Autor: Eivar Rojas

Diagrama de Clases

Ilustración **14**. Caso de Uso-Diseño 02 “Generar Captura de datos” (Diagrama de Clases)

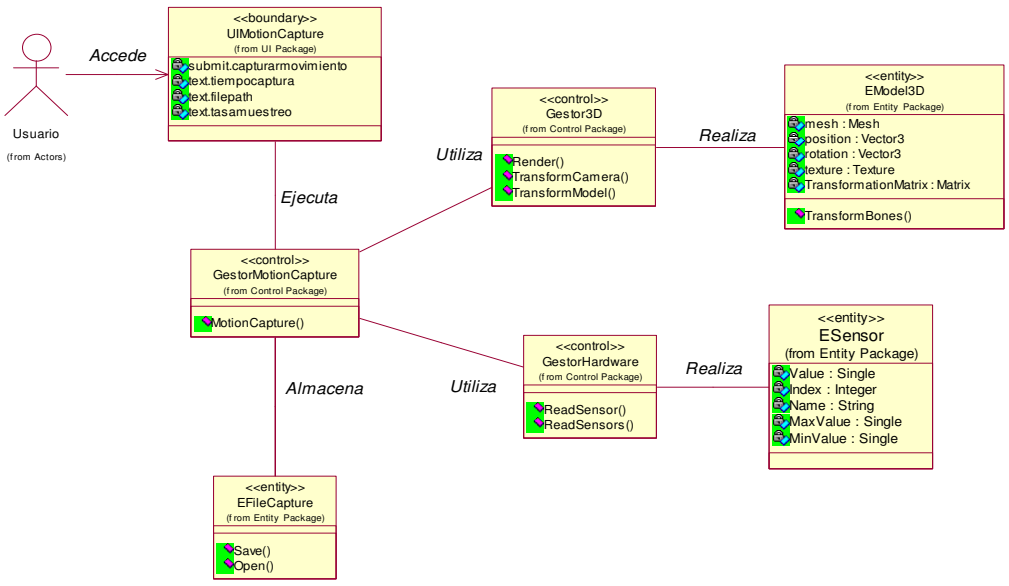
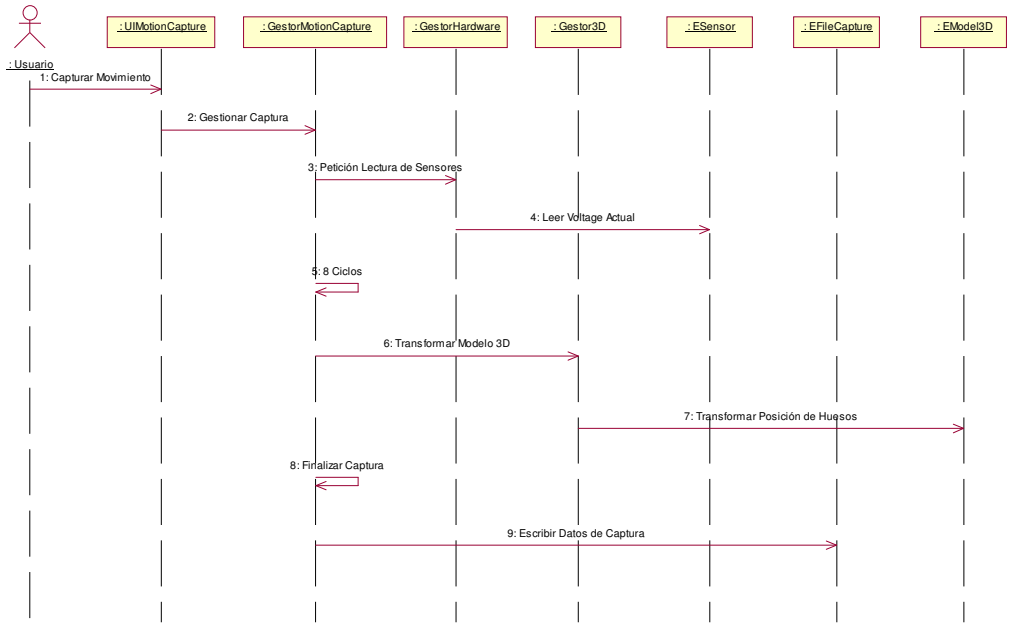


Diagrama de Interacción

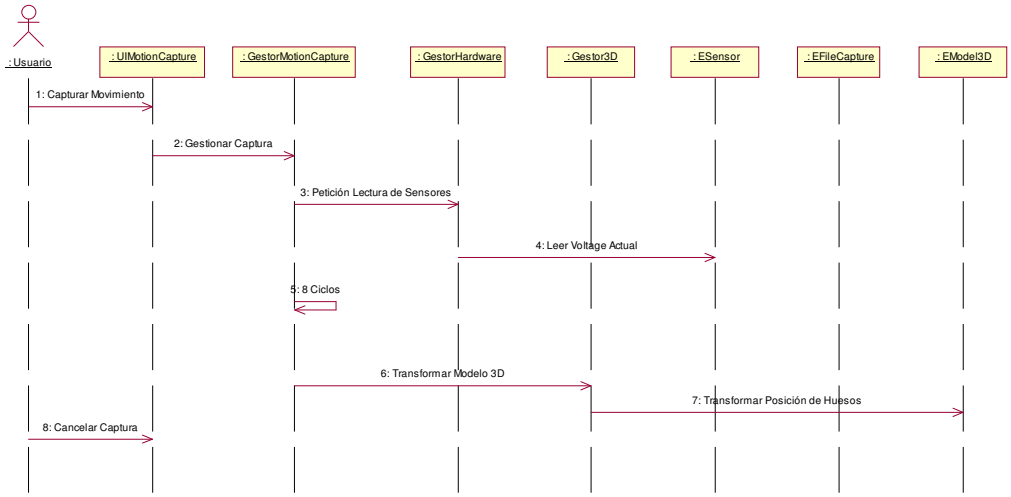
Ilustración **15**. Caso de Uso-Diseño 02 “Generar Captura de datos” (Diagrama de Interacción)



La ejecución es secuencial en el tiempo, comunicándose con las clases de control GestorMotionCapture, GestorHardware y Gestor3D con sus respectivas clases entidad Sensor, Model3d y FileCapture. Una vez el sistema Hardware se encuentra calibrado el usuario solicita realizar la captura de movimiento. El gestor de captura adopta los valores de configuración de la captura y la realiza durante el tiempo especificado. Almacenando los datos leídos durante la captura.

Flujo Alternativo

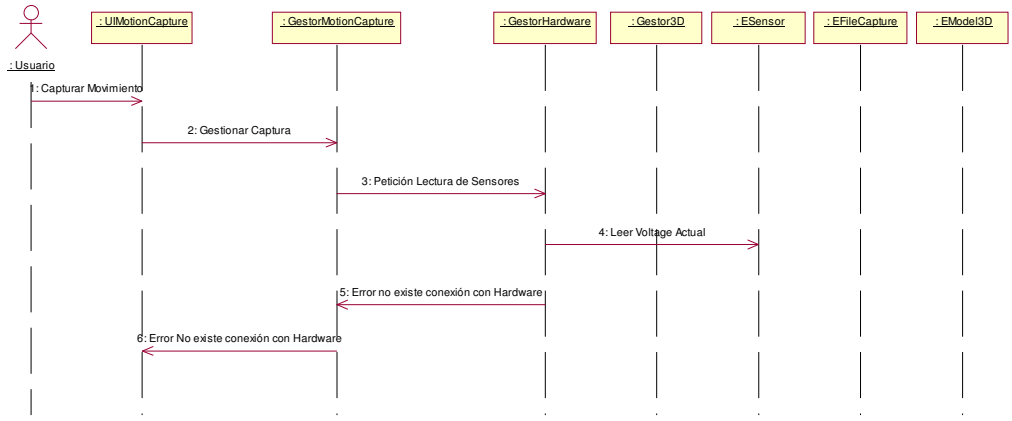
Ilustración **16**. Caso de Uso-Diseño 02 “Generar Captura de datos” Flujo Alternativo (Diagrama de Interacción)



La ejecución es secuencial en el tiempo, comunicándose con las clases de control GestorMotionCapture, GestorHardware y Gestor3D con sus respectivas clases entidad Sensor, Model3d y FileCapture. Una vez el sistema Hardware se encuentra calibrado el usuario solicita realizar la captura de movimiento. El gestor de captura adopta los valores de configuración de la captura y la realiza durante el tiempo especificado. Durante el proceso de captura el usuario aborta la operación. El sistema almacenará los datos capturados hasta el momento.

Flujo Alternativo

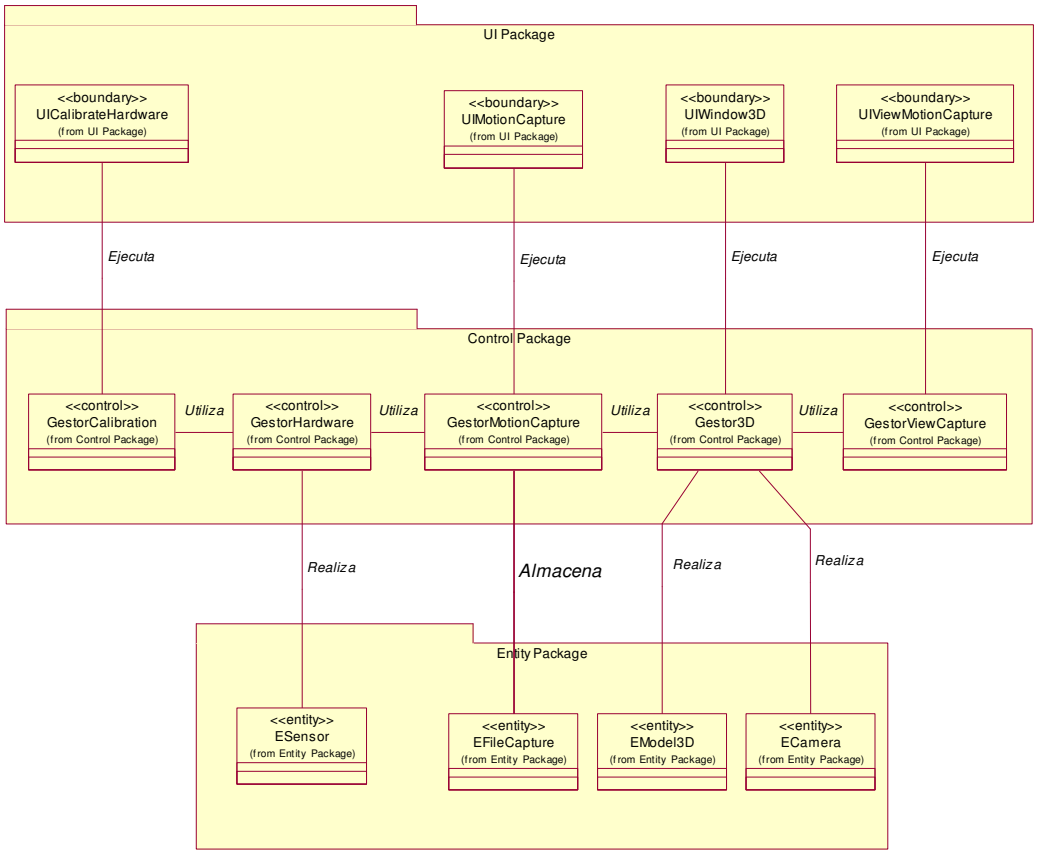
Ilustración **17**. Caso de Uso-Diseño 02 “Generar Captura de datos” Flujo Alternativo (Diagrama de Interacción)



La ejecución es secuencial en el tiempo, comunicándose con las clases de control GestorMotionCapture, GestorHardware y Gestor3D con sus respectivas clases entidad Sensor, Model3d y FileCapture. Una vez el sistema Hardware se encuentra calibrado el usuario solicita realizar la captura de movimiento. El gestor de captura adopta los valores de configuración de la captura y la realiza durante el tiempo especificado. Durante el proceso de captura el Gestor Hardware pierde conexión con el Sistema Hardware y decide abortar la operación. Almacenando los datos que se capturaron hasta el momento.

9.2. PAQUETES DE DISEÑO

Ilustración **18**. Paquetes de diseño



9.3. DESCRIPCION DE PAQUETES DE DISEÑO

Paquete: "UI Package"
Descripción:

Todas las clases son interfaces de usuario que confirman los requisitos especificados en la toma de requisitos.

Este paquete tiene todas las clases que van a dar como resultado, o que van a almacenar todos los datos necesarios para la creación de las interfaces por las que el usuario va a acceder a todas las funcionalidades que ofrece el sistema.

El paquete contendrá las siguientes clases:

- UICalibrateHardware, esta interfaz visualizará la pantalla de calibración del Sistema Hardware la cual contendrá todos los campos necesarios para que un usuario pueda realizar una correcta calibración. Una vez realizada la calibración correctamente el sistema quedará habilitado para realizar capturas de movimiento y se mostrará un listado con todos los sensores y sus valores respectivos de calibración.

- UIMotionCapture, a esta interfaz accederá el usuario una vez ya hecha la calibración del Sistema Hardware. Esta interfaz contendrá todos los campos necesarios para especificar el fichero de captura y los parámetros de captura.

- UIViewMotionCapture, a esta interfaz accederá el usuario cuando desee visualizar una captura previamente realizada. Esta interfaz tendrá los controles

estándar para reproducir una animación además el capo donde se especifica el fichero para visualizar.

- UIWindow3D, esta interfaz tendrá los controles necesarios para realizar modificaciones a la visualización del visor 3D, como alejar o acercar la ventana, rotar el mundo 3D y hacer paneo de la imagen.

Los métodos de las clases interfaz nos permitirán hacer llamadas a los métodos de las clases de control, que nos permitirán interactuar y visualizar con el mundo 3D. Además permitir la configuración del Sistema Hardware.

La creación de este paquete nos permitirá englobar todas las clases referidas a las clases de interfaz, con el fin de abstraer y agrupar la idea gráfica del a aplicación, siguiendo los requisitos especificados en las interfaces del capitulo de toma de requisitos.

Las clases de interfaz de usuario son independientes y se agrupan todas en el paquete UIPackage para seguir el patrón MVC (Model View Controller). El objetivo de agrupar todas las interfaces de usuario en un paquete, es permitir mayor reutilización en el desarrollo del proyecto debido a su alta modularidad.

Paquete: "Control"
Descripción:

- GestorCalibration, esta clase control el método que hace posible la calibración del Sistema hardware, además de mostrar los datos calibrados de el sistema Hardware.

- GestorMotionCapture, esta clase contiene todos los métodos necesarios para realizar la sincronización de la lectura de los sensores con la visualización tridimensional del proceso de captura. También la funcionalidad de guardar el archivo de captura de movimiento una vez realizada.

- GestorViewCapture, esta clase contiene los métodos que hacen posible la visualización de una captura de movimiento previamente realizada, sincronizando los datos leídos del archivo de captura y la visualización tridimensional de los movimientos.

- GestorHardware, esta clase contiene todos los métodos necesarios para realizar la comunicación del Sistema con el Sistema Hardware. Permitiendo realizar lecturas por sensor o lecturas por grupo de sensores. Además esta clase informará del estado de la conexión del Sistema Hardware.

- Gestor3D, es la clase que contiene los métodos necesarios para poder realizar la visualización del mundo, las transformaciones del mismo y transformar el sistema esquelético del modelo 3D.

Objetivo:

Las clases de control se encargan de la comunicación entre las clases de interfaz y las clases de entidad. Cuando una clase de control realiza varias llamadas, el método de la clase control lo que hará será realizar las llamadas oportunas a los métodos de las clases de entidad en el orden especificado en el diagrama de interacción.

El objetivo de creación de este paquete es englobar todo el proceso de la aplicación, es decir, poder tener en un solo paquete el funcionamiento del sistema de tal forma que nos facilite el acceso a todas sus funcionalidades, realizando dicho acceso de forma rápida, consistente e integra.

El tutorial descrito a continuación detalla una de las múltiples formas de generación de objetos tridimensionales en tiempo real. Además, el esbozo de la manipulación de geometrías mediante sistemas jerárquicos de huesos.

Toda el desarrollo se baza en la utilización de la API DirectX 9.0c, enfocando el desarrollo en la librería gráfica 3D (Direct3D). Para la utilización de las librerías 3D se va a utilizar Visual Studio .Net 2005. Teniendo claro que esta no es la única herramienta de generación de código en cuando a gráficos 3D se refiere. El lenguaje de programación es C# (C Sharp).

Un aspecto importante que hace parte del tutorial es la utilización de modelo 3D generado en un programa de generación de gráficos 3D (3D Studio Max) y el formato del archivo es MD5. Este formato de archivo brinda la posibilidad de obtener un objeto integrado por un sistema de huesos que es totalmente configurado en 3D Studio Max.

Para la generación de modelos MD5 existe un PLUG-IN que se integra a 3D Studio Max y permite la generación del archivo.

10. DESARROLLO DEL PROTOTIPO

El hardware del sistema está formado básicamente por un sistema de adquisición de datos, una etapa seguidor de voltaje y un grupo de sensores flexibles. El principio del sistema es sencillo, cada sensor es acoplado a una articulación del cuerpo, de tal forma que al doblarla sea posible medir su ángulo de flexión. La etapa seguidora de voltaje impide que el voltaje generado por el sensor baje a cero. Luego un sistema de adquisición de datos convierte los datos análogos en datos digitales y a través del bus de datos USB son enviados al computador.

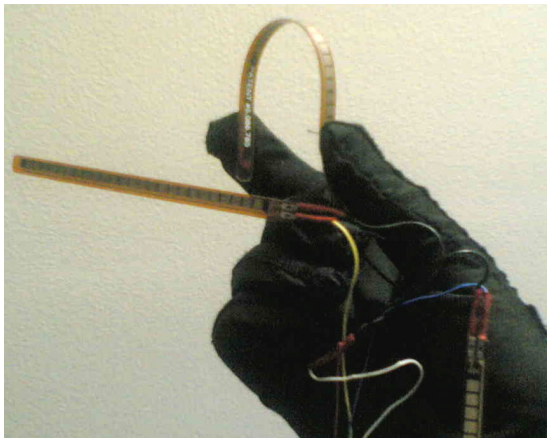
El sistema puede monitorear 8 sensores, a una razón de 1200 muestras por segundo. La información obtenida puede ser enviada 1.2 Mbps, esta velocidad de transferencia es mas que suficiente para captar aquellos movimientos que son perceptibles a la vista. El software del sistema utiliza la información capturada para animar un modelo de huesos del brazo 3D. Esta animación muestra un movimiento suave y tiene un retardo mínimo de (0.05 segundos) entre la ejecución del movimiento y su representación gráfica.

10.1. DESCRIPCIÓN

El hardware del prototipo esta formado por cuatro etapas: un conjunto de sensores que detectan el movimiento de las articulaciones, un circuito que permite estabilizar el voltaje generado y un sistema de adquisición de datos para digitalizar las señales de estos sensores y una aplicación encargada de adecuar y visualizar la información.

Los sensores están hechos de un polímero flexible y permiten medir la magnitud de la deformación a que son sometidos. En caso de que el sensor sea doblado, su resistencia varía y puede ser relacionado con el ángulo de flexión. Si se acopla uno de estos sensores a una articulación y esta se dobla deformando el sensor, entonces es posible registrar su movimiento. (Ver ilustración 10.1 19)

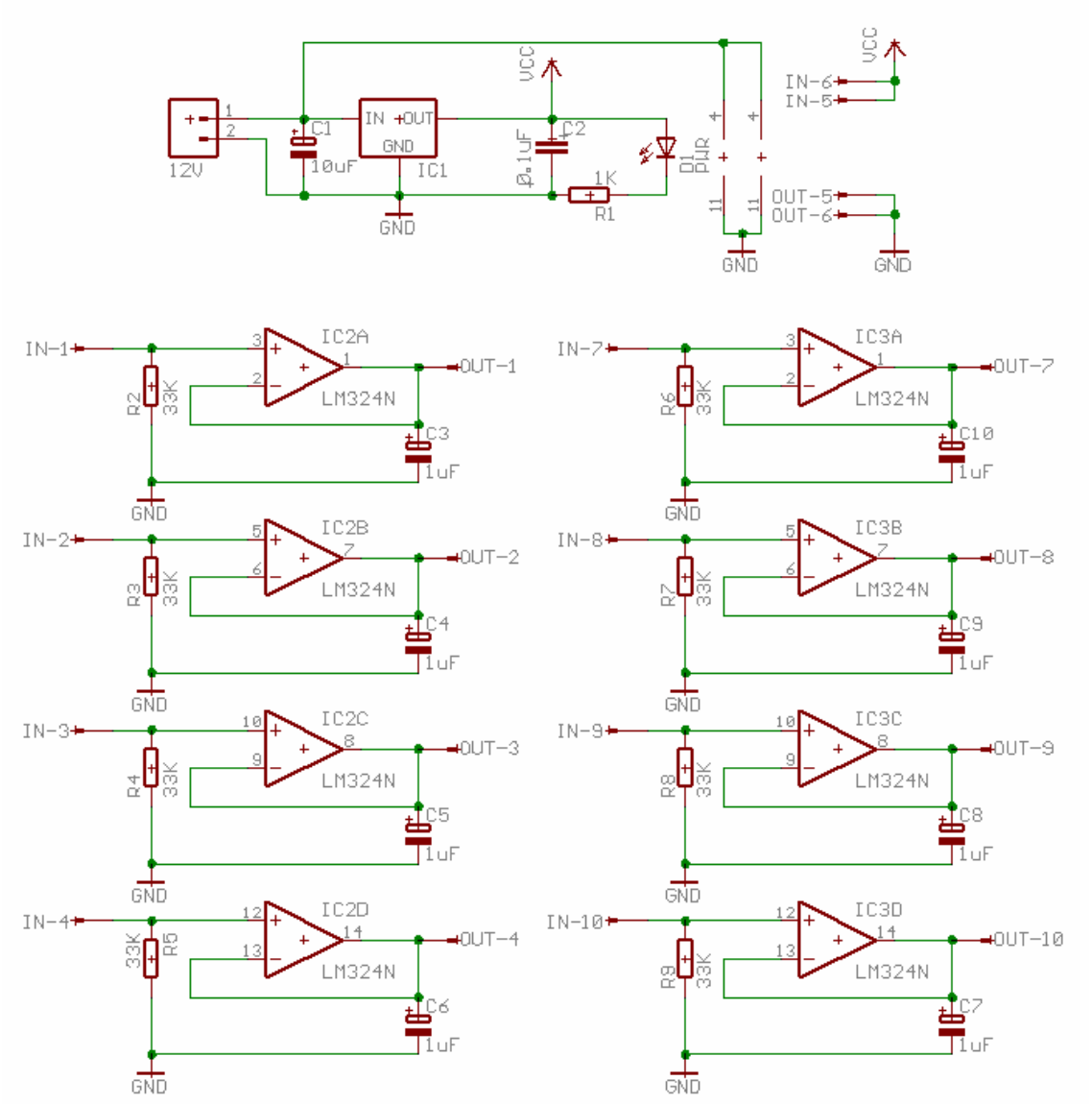
Ilustración **19**. Sensores flexibles



El circuito seguidor de voltaje evita que la señal generada por los sensores pierdan su magnitud en la medida, aumentando la impedancia de entrada. En resumen el

circuito es un arreglo de amplificadores operacionales que funcionan como seguidores de voltaje sin invertir polaridad o amplificar la señal. (Ver ilustración)

Ilustración **20**. Diagrama esquemático del circuito seguidor de voltaje.



El sistema de captura permite capturar la información de 8 sensores, la magnitud del voltaje a la salida del circuito seguidor de voltaje y enviada a la computadora por medio de puerto USB. Los valores capturados son interpretados por el sistema de adquisición en una resolución de 12 bits, permitiendo gran sensibilidad en cada dato capturado. El sistema de adquisición entrega los datos uno por uno debido a que el puerto es serial. El diagrama de bloques del sistema de adquisición puede ser visto en la ilustración número 4.2.6

Cada dato recibido es interpretado como un ángulo se debe aplicar una transformación lineal, la función de transferencia se encuentra de la siguiente forma:

Vmax = Máximo voltaje generado por el punto de inflexión.
Vmin = Mínimo voltaje generado por el punto de inflexión.
Amax = Angulo máximo soportado por el punto de inflexión.
Amin = Angulo mínimo soportado por el punto de inflexión.

Arang = Amax – Amin.
Vrang = Vmax – Vmin.
V = Voltaje capturado.

$$AnguloFelxión = \frac{Avrang * V}{Vrang} + A\min$$

V

El software del sistema es ejecutado en una computadora externa y permite leer la información proveniente del hardware de captura. El comportamiento de los

capturados puede visualizarse directamente en la aplicación mediante un brazo tridimensional que a su interior se encuentra una estructura de huesos.

Para ingresar un modelo tridimensional a la aplicación éste debe contar con una estructura de huesos y con una nomenclatura especial para sus nombres. En este caso la aplicación utilizada para crear el brazo es 3D Studio Max V. 7.0, dentro de esta aplicación existe una herramienta llamada Skin que permite asignar un modelo esquelético a un modelo tridimensional e indicar como cada hueso afecta el modelo tridimensional .(Ver Figura 10.1.21).

Ilustración **21**. Modelo tridimensional

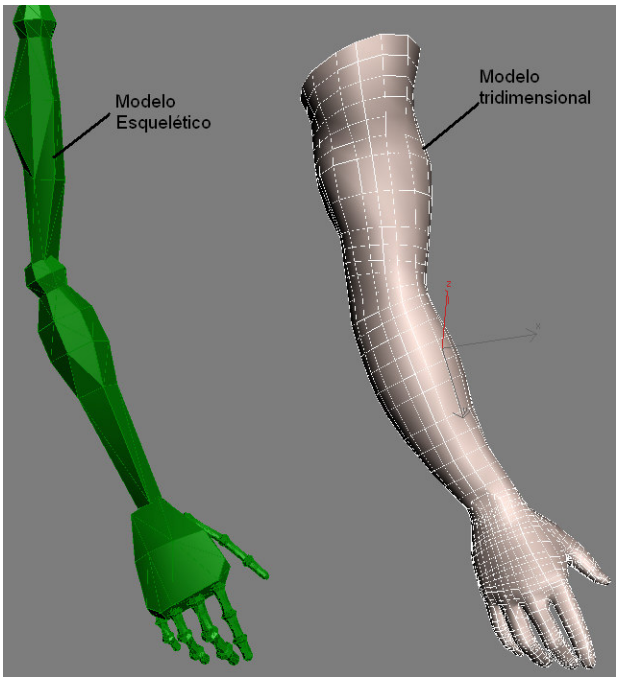
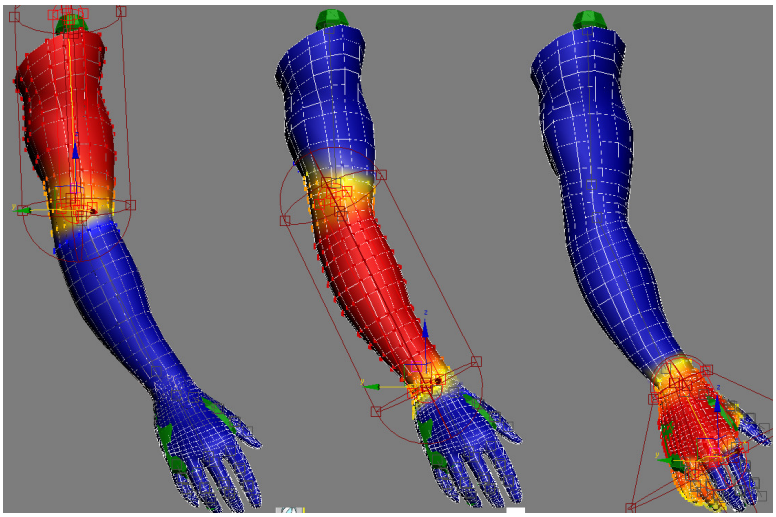


Ilustración **22**. Herramienta Skin



Cuando el modelo cuenta con un Skin bien realizado se procede a exportar el modelo tridimensional, existe una herramienta denominada MD5Export (Ver Ilustración 10.1.23) que permite generar un archivo de tipo MD5 del brazo y es que se va a interpretar dentro de la aplicación el modelo debe tener una característica y es que dentro de 3D Studio Max éste debe ser un MeshEditable. Y cada hueso debe contar con los siguientes nombres:

SCM_Brazo.
SCM_Antebrazo.
SCM_Muneca.
SCM_Pulgar_1, SCM_Pulgar_2, SCM_Pulgar_3.
SCM_Indice_1, SCM_Indice_2, SCM_Indice_3.
SCM_Corazon_1, SCM_Corazon_2, SCM_Corazon_3

En los anexos, dentro del tutorial de Direct3D se puede ver en detalle como se trabaja con un modelo MD5.

Ilustración **23**. Herramienta para exportar

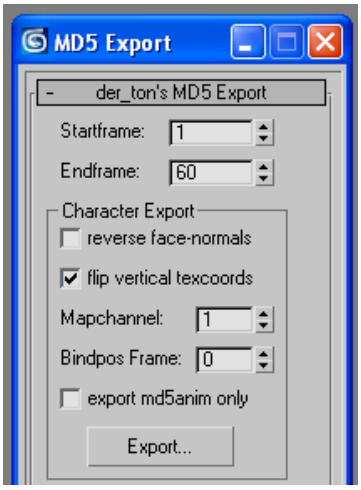
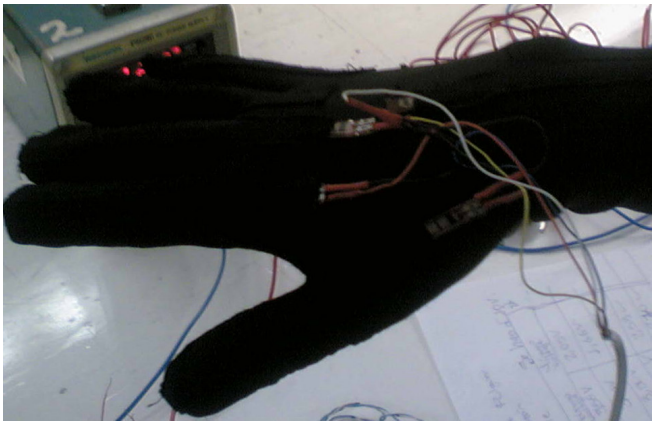
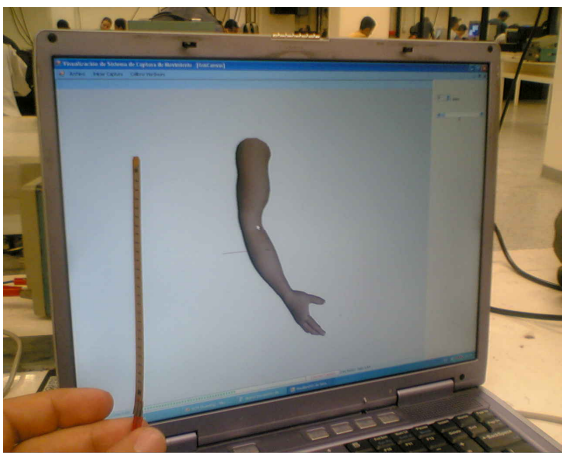
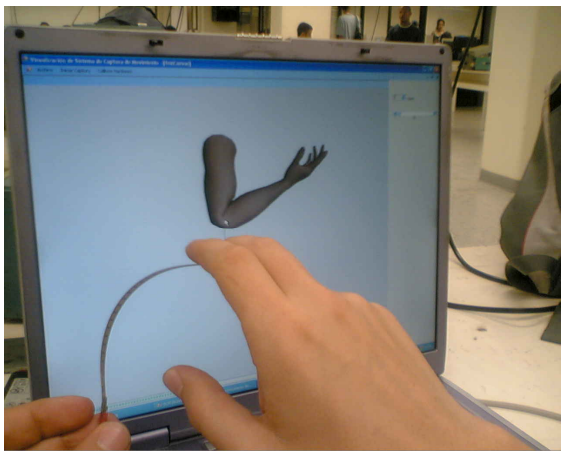


Ilustración **24**. Imágenes del Prototipo de captura







11. CONCLUSIONES

- El sistema desarrollado brinda una gran ventaja frente a otros sistemas mecánicos de captura, logrando que el tamaño de las partes finales de adquisición sean muy pequeñas (los sensores), ayudando a la portabilidad.

- El desarrollo de este prototipo insita a nuevas personas a generar soluciones que puedan ser implementadas para solucionar problemas específicos, tal como se propone en la sección de trabajo futuro.

- Con este prototipo implementado en el campo de animación tridimensional, se logrará disminuir el tiempo de animación y aumentar la calidad de la misma.

- El enfoque de desarrollo y la metodología utilizada para el desarrollo de este proyecto, permitió una adecuada organización y seguimiento del trabajo realizado, con el valor añadido, de permitir ampliar el conocimiento en esta área poco explorada en nuestro entorno.

- Gracias a que los componentes utilizados para el desarrollo del sistema son de bajo costo, muchas empresas pequeñas podrían tener acceso a este tipo de tecnología.

12. TRABAJO FUTURO

- Incrementar al sistema el número de sensores de captura permitiendo la captura de otras extremidades del cuerpo. Además, un sensor que describa la posición del cuerpo para poder realizar desplazamientos.

- Experimentar con otro tipo modelos tridimensionales, para generar más implementaciones en torno al tema de captura de movimiento. Las implementaciones pueden ser animación de personajes, títeres virtuales.

- Una implementación importante que se puede desarrollar para complementar este prototipo es la creación de una aplicación más compleja donde se pueda integrar la interacción del usuario con el entorno tridimensional, permitiendo la manipulación de objetos virtuales en tiempo real.

- Generar un sistema donde se pueda interpretar los gestos realizados por una persona impedida del habla.

BIBLIOGRAFÍA

ADDISON, Wesley. Introduction to computer Graphics [en línea]. Ottawa: University of Ottawa, 2005. [Consultada 23 de junio de 2004]. Disponible en Internet:http://biology.ncsa.uiuc.edu/cgin/infosrch.cgi?cmd=getdoc&coll=0650&db=bks&fname=/SGI_Developer/OpenGL_RM/index.html.

ALAN, Akin. Microsoft and 3D Graphics: A Case Study in Suppressing [en línea]. California: Moral Highground Productions, 1998. [Consultada 12 de junio de 2004]. Disponible en Internet: http://www.vcnet.com/bms.

HAWKINS, Kevin. What is DirectX [en línea]. Salt Lake City: Myopic Rhino games, 1999. [Consultada 13 de Junio de 2004]. Disponible en internet: http://www.gamedev.net.

HERDA, Louis. Skeleton-Based Motion Capture for Robust Reconstruction of Human Motion [en línea]. Lausanne: Computer Graphics Lab EPFL, Switzerland, 2000. [Consultada 23 de Junio de 2001]. Disponible en internet: http://www.gamedev.net/MotionCapture.html.

KESSLER, G.Drew. Evaluation of the CyberGlove as a Whole Hand Input Device [en línea]. Georgia: Graphics, Visualization & Usability Center. Georgia Institute of Technology, 2002. [Consultada 13 de Agosto de 2005]. Disponible en Internet: http://www.immersion.com/cyberblove.htm.

SERPA VIEIRA DA SILVA, Fernando Wagner. Sistema de Animación Basado en el movimiento capturado [en línea]. Rio de Janeiro: Laboratorio de computación

gráfica, COPPE-Sistemas/URFJ, 2000. [Consultada 28 de Mayo de 2004]. Disponible en Internet: http://www.visgraf.impa.br/mocap/tese.

SERPA VIEIRA DA SILVA, Fernando Wagner. Reparametrización de un sistema de captura [en línea]. Rio de Janeiro: Laboratorio de computación gráfica, COPPE-Sistemas/URFJ, 2000. [Consultada 28 de Mayo de 2004]. Disponible en Internet: http://www.visgraf.impa.br/mocap/tese.

Timeline of CGI in films [en línea]. Florida: Wikipedia foundation, 2005. (Consultada 23 de junio de 2004). Disponible en: http://en.wikipedia.org/wiki/time_of_CGI_in_films.htm.

ANEXOS

Anexo 1. Descripción de casos de uso

Número: 001

Nombre de Caso de Uso: "Calibrar y Configurar el sistema Hardware de captura"

Actor(es): Usuario del Sistema

Descripción:

Este caso de uso describe como el usuario podrá hacer la calibración del sistema de adquisición de datos, de éste depende el óptimo desempeño del sistema software.

Flujo de Eventos	
Curso normal	Alternativas
El usuario selecciona la opción de calibración del sistema hardware.	
El sistema mediante ventanas guía al usuario para la calibración de sensor por sensor, solicitando al usuario generar inflexión en la articulación correspondiente, en dos pasos, inflexión máxima e inflexión mínima.	2.1 El usuario selecciona el comando Cancelar. sale de la etapa de calibración 2.2 El sistema muestra la advertencia en pantalla "la aplicación no puede continuar si no se calibran todos los sensores". La aplicación se cierra. 2.3 Pasa al caso de uso 001 desde el primer paso.
El sistema Software ajusta los datos recibidos, para determinar los puntos máximos y mínimos de inflexión de las articulaciones del brazo 3D.	3.1. Alguno de los sensores no cumplió con la variación mínima. 3.2. Es sistema muestra el mensaje "El sensor ubicado en la posición X no cumplió con los Requisitos mínimos de variación, verifique su conexión". La aplicación finaliza, se deben corroborar las conexiones de los sensores 3.3. Se ha perdido conexión con el Sistema

	Hardware. 3.4. El sistema muestra el mensaje “El sistema ha perdido conexión con el Sistema Hardware no se puede continuar con la calibración”. La aplicación finaliza.
Se muestra en pantalla el reporte de los datos de calibración.	

Requisitos Especiales

El sistema Hardware debe estar correctamente conectado al PC por medio del puerto USB.

Pre-Condiciones

Conexión del los sensores al sistema de adquisición.

Post-Condiciones

N/A

Puntos de Extensión

N/A

Número: 002

Nombre de Caso de Uso: "Generar Captura de Datos"

Actor(es): Usuario Sistema

Descripción:

Este caso de uso describe como el usuario podrá hacer una captura de movimiento.

Flujo de Eventos	
Curso normal	Alternativas
El caso de uso inicia cuando el usuario accede a la aplicación.	
Hace la respectiva calibración del sistema Hardware.	
El usuario accede al comando de captura de movimiento	
El sistema solicita que se seleccione el fichero para guardar el archivo el cual contendrá la información capturada.	
El sistema solicita especificar el tiempo durante el cual se va a hacer la captura de movimiento, y el número de muestras por segundo que realizará el sistema de adquisición de datos.	5.1 El usuario decide Cancelar el proceso de Captura de Movimiento. 5.2 Pasa al caso de uso 001 desde el primer paso.
El sistema realiza la captura durante el tiempo especificado, y el sistema representa la información capturada por un brazo presente en el visualizador 3D el cual reacciona a los movimientos del brazo humano.	La aplicación software perdió conexión con el sistema Hardware, no se pueden hacer mas captura de datos, se guarda solo lo capturado hasta el momento. El sistema muestra en pantalla el mensaje "No existe conexión entre el sistema Software y el sistema Hardware". 6.2 Pasa al caso de uso 001 desde el primer paso.
El sistema cumple con el tiempo establecido y da por finalizada la captura. Muestra el mensaje "Se ha finalizado la captura, Desea visualizar los movimientos capturados" y se presentan dos opciones Aceptar o Cancelar.	7.1 la visualización 3D finaliza.
El sistema pasa al caso de uso Número 5 paso Número 4	

Requisitos Especiales

El sistema Hardware debe estar correctamente conectado al PC por medio del puerto USB.

Pre-Condiciones

El usuario debe calibrar el sistema hardware

Post-Condiciones

N/A

Puntos de Extensión

CU01

Número: 003

Nombre de Caso de Uso: "Manipulación del Visor 3D"

Actor(es): Usuario Sistema

Descripción:

Este caso de uso describe como el usuario podrá manipular el visualizador 3D.

Flujo de Eventos	
Curso normal	Alternativas
El caso de uso inicia cuando el usuario accede a la aplicación.	
Realiza el caso de uso 003 desde el paso 2.	
El usuario habilita los comandos de manipulación del visor, representados por Zoom, Pan y Rotación.	
El usuario deshabilita los comandos oprimiendo la tecla Escape del teclado.	

Requisitos Especiales

Pre-Condiciones

Debe existir como mínimo un objeto geométrico

Post-Condiciones

N/A

Puntos de Extensión

N/A

Número: 004

Nombre de Caso de Uso: "Reproducir Captura de Movimiento"

Actor(es): Usuario Sistema

Descripción:

Este caso de uso describe como el usuario podrá hacer una reproducción de la captura movimiento.

Flujo de Eventos	
Curso normal	Alternativas
El caso de uso inicia cuando el usuario accede a la aplicación.	
El usuario habilita el comando de ejecución de captura	
El sistema solicita la selección de un archivo para reproducir una captura hecha previamente.	El tipo de archivo es incompatible con el predefinido. Retorna al paso número 1 de este caso de uso.
El sistema inicia la visualización 3D con el brazo virtual.	
El sistema hace la reproducción de los datos almacenados en el archivo y entra en un ciclo infinito la animación.	5.1 El usuario detiene la reproducción de la animación.

Requisitos Especiales

Se deben seleccionar extensiones de archivo predefinidas en la aplicación

Pre-Condiciones

N/A

Post-Condiciones

N/A

Puntos de Extensión

N/A

Anexo 2. Modelos de arquitectura

Los casos de uso que se describen a continuación son necesarios para que la aplicación pueda ser en gran parte funcional y se pueda hacer captura de movimiento. Estos casos de uso son iniciados por el usuario y están descritos a continuación:

- Calibrar y Configurar el sistema Hardware de captura
- Hacer Captura de Datos

Número: 001

Nombre de Caso de Uso: "Calibrar y Configurar el sistema Hardware de captura"

Ilustración **25**. Caso de Uso-Requisitos 01 "Calibrar y Configurar el sistema Hardware de Captura"

Los flujos de este caso de uso son realizados siempre y cuando el usuario inicie la aplicación. Esta calibración se deberá hacer a través de una especie de "wizard" que guiará al usuario en todo el proceso. Esta calibración es muy importante, pues los datos recolectados en esta fase serán los que establezcan los máximos y los mínimos, de los puntos de inflexión. Esto significa que si no se realiza la mínima y

la máxima inflexión de cada sensor, los movimientos capturados y visualizados serían mal descritos por la aplicación software.

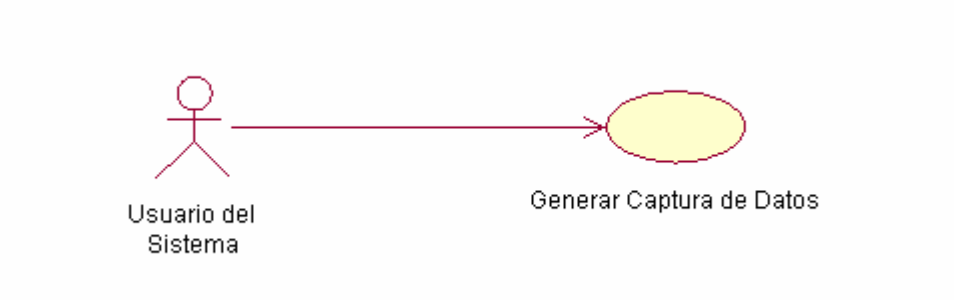
Se deben calibrar 8 sensores que se distribuyen en el brazo de la siguiente manera: un sensor en el dedo pulgar, uno en el dedo índice, uno en el dedo anular, uno en la muñeca, uno en el antebrazo, uno en el codo y dos en el hombro.

Al calibrar correctamente los sensores, la aplicación podrá representar adecuadamente los movimientos que se ejecutan en el mundo real.

Número: 002

Nombre de Caso de Uso: "Generar Captura de Datos"

Ilustración **26**. Caso de Uso-Requisitos 02 "Generar Captura de Datos"

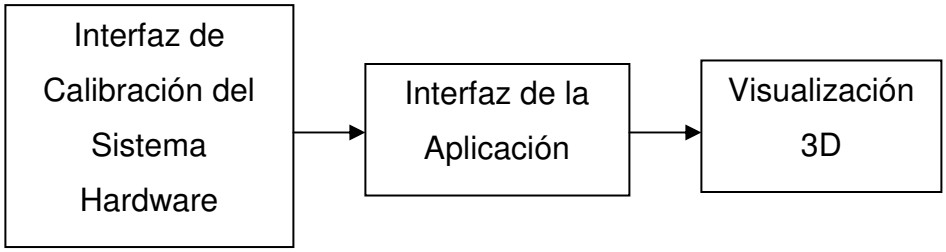


El caso de uso inicia cuando el usuario después de haber calibrado el sistema hardware (Caso de uso 001) procede a hacer la captura de movimiento. Primero se debe definir el fichero donde serán guardados los datos capturados, este tipo de archivo es un archivo que, a su vez debe ser interpretado por otra aplicación tridimensional (3D Studio Max) para análisis o visualización de la captura. Luego se establecen dos parámetros importantes y necesarios para la captura: el tiempo de captura y el número de muestras por segundo que efectuará el sistema hardware. Una vez introducidos los parámetros, la aplicación mediante el visualizador representa en un brazo tridimensional los movimientos efectuados en el brazo real hasta que se cumpla el tiempo establecido para la captura.

INTERFACES

A continuación un primer acercamiento de las interfaces que interactúan con el usuario:

Ilustración **27**. Interfaces de la aplicación



Calibración del Sistema Hardware: Esta ventana tendrá tres regiones:

- Diagrama del sensor a calibrar, en esta región se ubicará un gráfico que represente como se debe calibrar el sensor específico.

- Datos del sensor calibrado, esta región publicará los valores de los voltajes máximo y mínimo que genere cada sensor cuando se calibra.

- Reporte de Calibración, al finalizar la calibración se publicará un reporte de los datos calibrados discriminados por cada sensor.

- Interfaz de la Aplicación: estará dotada de todas las funcionalidades del sistema y consta de las siguientes áreas:

- Barra de menú, desde esta barra se podrá acceder a todas la herramientas que posee la aplicación.

- Herramientas de Visualización, se ubicarán todos los comandos que permitan la manipulación del visor 3D, (Zoom, Pan, Rotar).

- Reproducción de Capturas, se incluirán comandos para reproducir una captura previamente generada, (Play, Stop, Pausa).

- Barra de Herramientas, esta barra tendrá ubicada todos los comandos para generar captura, calibrar, generar interacción.

- Solapa de Objetos, estarán ubicados comandos para ingresar objetos geométricos 3D.

- Barra de transformación, estará dotada de comandos para manipular los objetos geométricos, (Rotar y Mover).

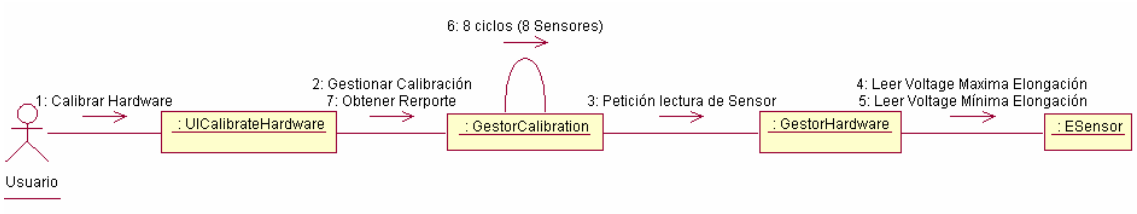
- Visualización 3D: Encargada de la visualización tridimensional de las capturas y de de los objetos geométricos.

Anexo 3. Desarrollo del sistema software (Análisis)

Número: 001

Nombre de Caso de Uso-Análisis: “Calibrar Sistema Hardware de Captura”

Ilustración **28**. Caso de Uso-Análisis 01 “Calibrar Sistema Hardware de Captura”



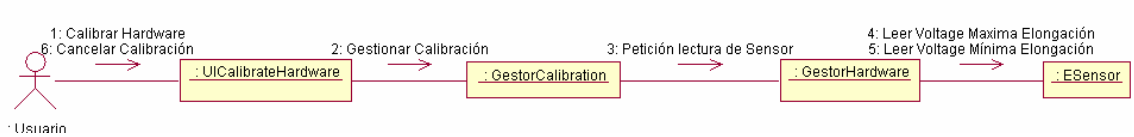
Definición

El Usuario inicia el caso de uso interactuando con la interfaz de usuario, que le permitirá pulsar el botón de Calibrar el Sistema Hardware. Esta interfaz de usuario se comunicará con la clase control proporcionándole la orden pulsada, esta clase control invocará la clase control solicitando lectura del sensor. Al existir múltiples sensores la clase control invocara la lectura de valores ocho veces involucrando al usuario en un proceso interactivo donde se presentará mediante la interfaz de usuario, gráficos que representen como se debe calibrar cada sensor. Al finalizar la calibración la interfaz de usuario presenta un reporte con los datos adoptados en la calibración.

Flujo alternativo

El Usuario cancela el proceso de calibración del Sistema Hardware.

Ilustración **29**. Caso de Uso-Análisis 01 “Calibrar Sistema Hardware de Captura”

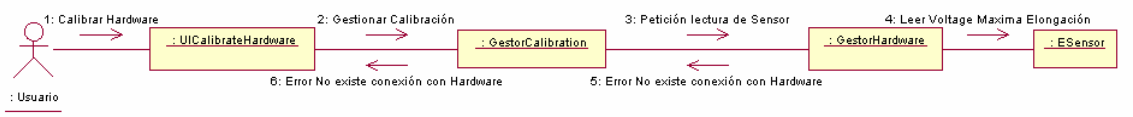


El Usuario inicia el caso de uso interactuando con la interfaz de usuario, que le permitirá pulsar el botón de Calibrar el Sistema Hardware. Esta interfaz de usuario se comunicará con la clase control proporcionándole la orden pulsada, esta clase control invocará la clase control encargada de gestionar el sistema Hardware, solicitando lectura del sensor. Al existir múltiples sensores la clase control invocara la lectura de valores ocho veces involucrando al usuario en un proceso interactivo donde se presentará mediante la interfaz de usuario, gráficos que representen como se debe calibrar cada sensor. El usuario cancela el proceso de calibración. La interfaz de usuario comunicará al usuario que la calibración no fue realizada de manera correcta y no se adoptaran los datos leídos.

Flujo alternativo

La aplicación pierde conexión con el Sistema Hardware.

Ilustración **30**. Caso de Uso-Análisis 01 “Calibrar Sistema Hardware de Captura” (Flujo Alternativo)

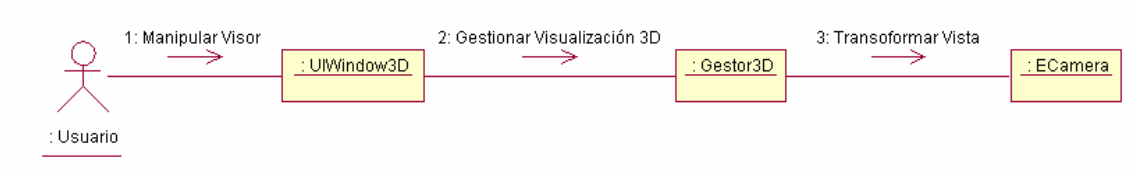


El Usuario inicia el caso de uso interactuando con la interfaz de usuario, que le permitirá pulsar el botón de Calibrar el Sistema Hardware. Esta interfaz de usuario se comunicará con la clase control proporcionándole la orden pulsada, esta clase control invocará la clase control encargada de gestionar el sistema Hardware, solicitando lectura del sensor. Al existir múltiples sensores la clase control invocara la lectura de valores ocho veces involucrando al usuario en un proceso interactivo donde se presentará mediante la interfaz de usuario, gráficos que representen como se debe calibrar cada sensor. Durante la calibración de cada sensor la clase que gestiona el Hardware pierde conexión con el Sistema emitiendo un mensaje de error al la clase control de calibración. La clase control determina no continuar con el procedimiento y comunica al usuario mediante la interfaz de usuario la falla de conexión con el Sistema Hardware y la imposibilidad de continuar con el proceso solicitando verificación de la conexión del hardware.

Número: 003

Nombre de Caso de Uso-Análisis: "Manipulación Visor 3D"

Ilustración **31**. Caso de Uso-Análisis 03 "Manipulación visor 3D"



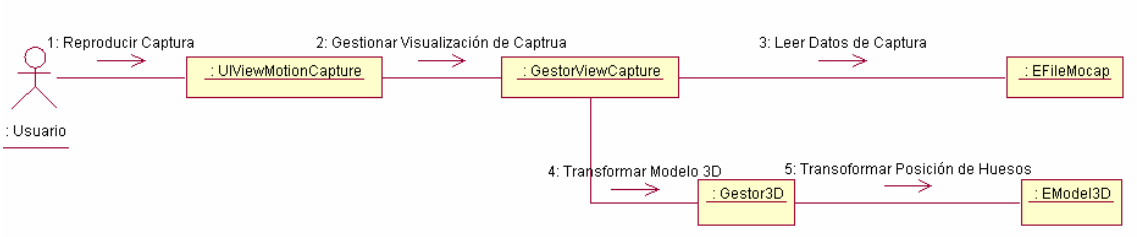
Definición

El Usuario inicia el caso de uso interactuando con la interfaz de usuario que le permitirá pulsar cualquiera de los tres botones los cuales tendrán involucrados tres tipos de transformación, Rotar vista, Acercar-Alejar vista, Paneo de vista. Ésta interfaz solicitará se comunicará con la clase control proporcionándole la orden pulsada. Mediante la interacción del usuario con el Mouse dentro de la ventana de visualización transformará el espacio de visualización.

Número: 004

Nombre de Caso de Uso-Análisis: "Reproducir Captura de Movimiento"

Ilustración **32**. Caso de Uso-Análisis 04 "Reproducir Captura de movimiento"



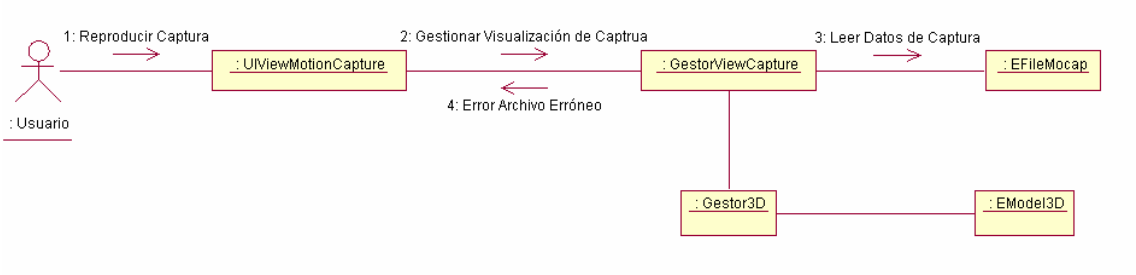
Definición

El Usuario inicia el caso de uso interactuando con la interfaz de usuario, seleccionando la opción de Reproducir Captura de Movimiento. Esta interfaz de usuario se comunicará con la clase control proporcionándole la orden pulsada, la clase control invocará la lectura de los datos del archivo y la visualización de la animación. La animación se reproducirá en un bucle infinito y solo finalizara hasta que el usuario pare la animación.

Flujo alternativo

El tipo de archivo es incompatible.

Ilustración **33**. Caso de Uso-Análisis 04 “Reproducir Captura de movimiento” Flujo Alternativo



El Usuario inicia el caso de uso interactuando con la interfaz de usuario, seleccionando la opción de Reproducir Captura de Movimiento. Esta interfaz de usuario se comunicará con la clase control proporcionándole la orden pulsada, la clase control invocará la lectura de los datos del archivo y la visualización de la animación. En el proceso de lectura del archivo se genera un error que puede se producido por incompatibilidad o errores en el archivo, la clase control finaliza el proceso de reproducción de la animación y notifica mediante la interfase de usuario el número línea del archivo de captura donde se generó la inconsistencia.

Anexo 4. DESARROLLO DEL SISTEMA SOFTWARE (DISEÑO)

Número: 001

Nombre de Caso de Uso-Diseño: “Calibrar Sistema Hardware de Captura”

Estado: (Liberad/ En Revisión/ En Desarrollo)

Autor: Eivar Rojas

Diagrama de Clases

Ilustración **34**. Caso de Uso-Diseño 01 “Calibrar Sistema Hardware de Captura” (Diagrama de Clases)

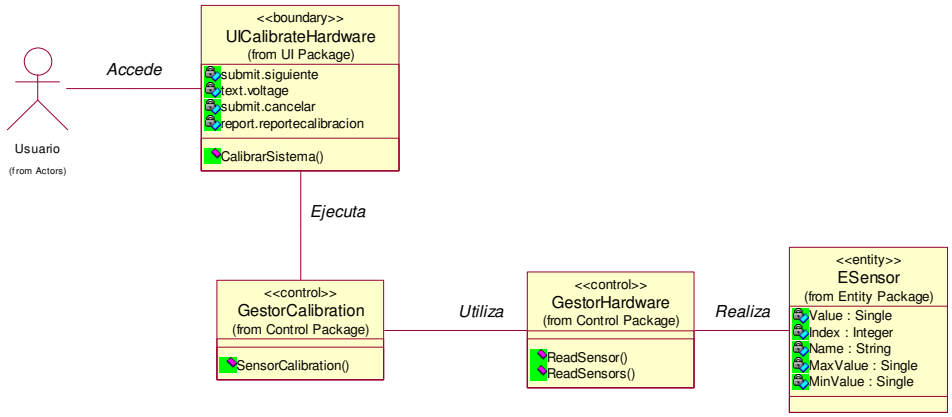
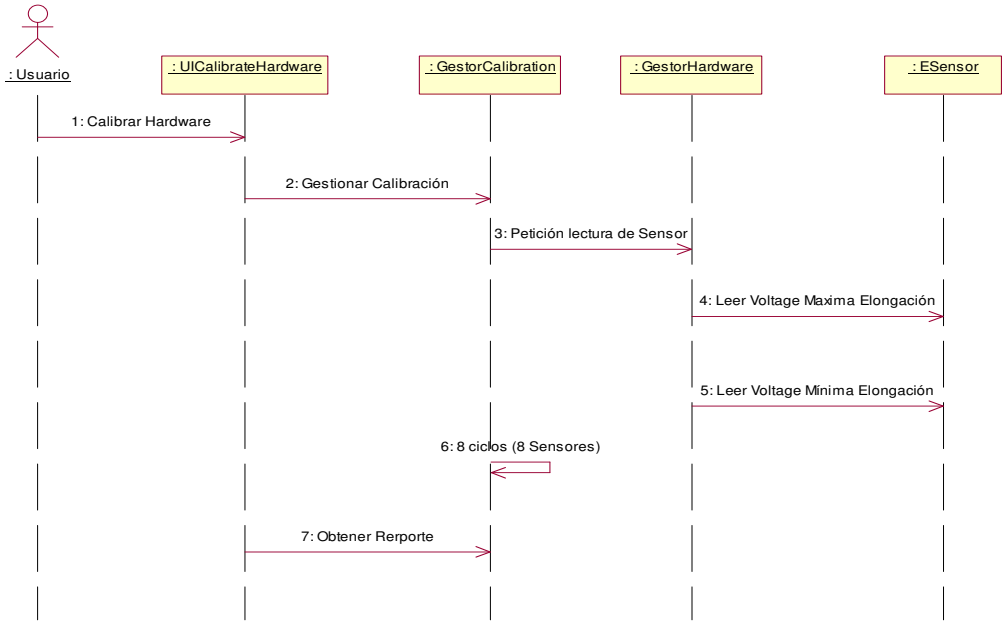


Diagrama de Interacción

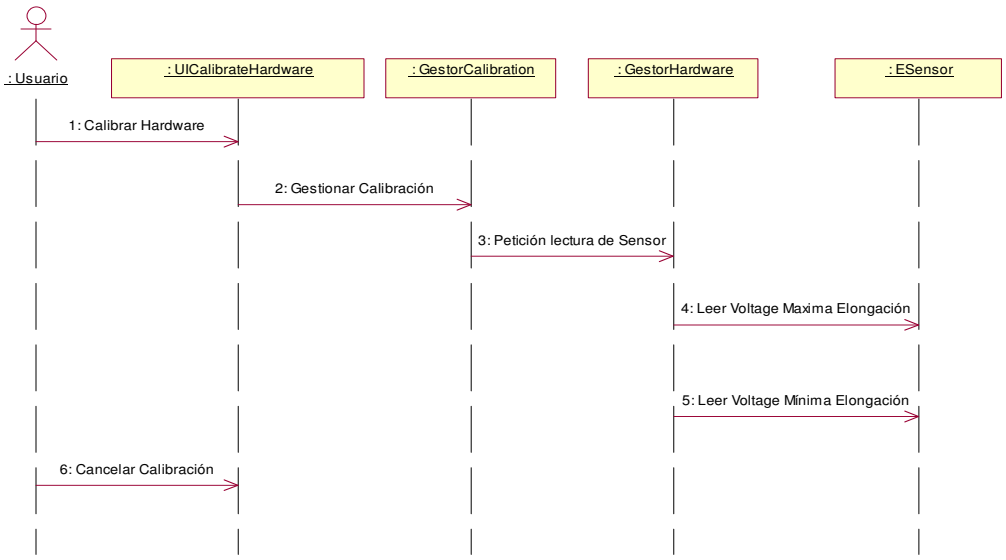
Ilustración **35**. Caso de Uso-Diseño 01 “Calibrar Sistema Hardware de Captura” (Diagrama de Interacción)



La ejecución es secuencial en el tiempo, comunicándose con las clases de control GestorCalibración y GestorHardware, con su respectiva clase entidad Sensor. El usuario solicita realizar la calibración del Sistema Hardware. El gestor de calibración transmite la petición y procede a realizar la lectura de los 8 sensores. Finalizando con un informe de los 8 sensores con sus respectivas lecturas.

Flujo Alternativo

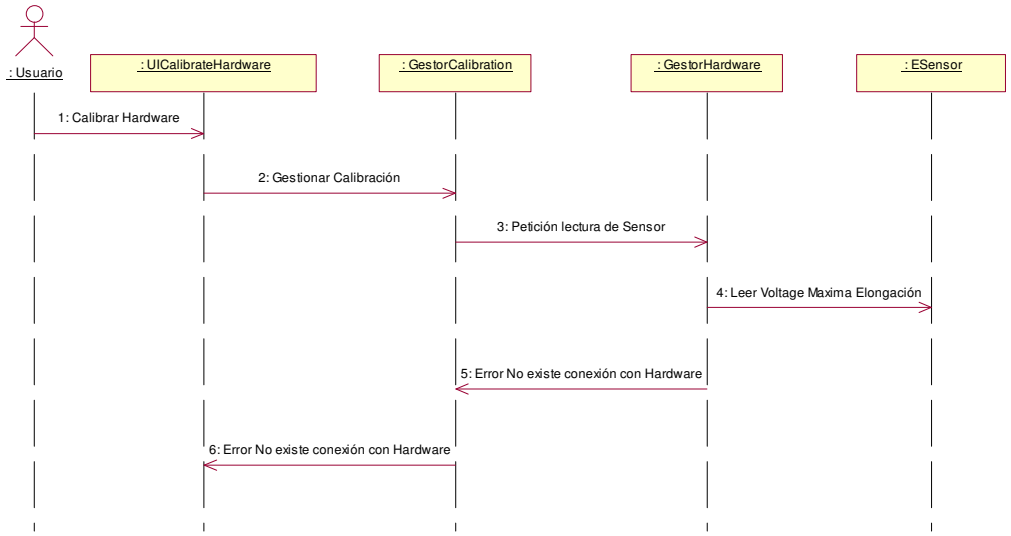
Ilustración **36**. Caso de Uso-Diseño 01 “Calibrar Sistema Hardware de Captura” Flujo Alternativo (Diagrama de Interacción)



La ejecución es secuencial en el tiempo, comunicándose con las clases de control GestorCalibración y GestorHardware, con su respectiva clase entidad Sensor. El usuario solicita realizar la calibración del Sistema Hardware. El gestor de calibración transmite la petición y procede a realizar la lectura de los 8 sensores. En el proceso de calibración de cada sensor el usuario aborta la operación. El sistema no adoptara los datos capturados como valores válidos de calibración.

Flujo Alternativo

Ilustración **37**. Caso de Uso-Diseño 01 “Calibrar Sistema Hardware de Captura” Flujo Alternativo (Diagrama de Interacción)



La ejecución es secuencial en el tiempo, comunicándose con las clases de control GestorCalibración y GestorHardware, con su respectiva clase entidad Sensor. El usuario solicita realizar la calibración del Sistema Hardware. El gestor de calibración transmite la petición y procede a realizar la lectura de los 8 sensores. En el proceso de calibración de cada sensor el Gestor Hardware pierde conexión con el Sistema Hardware y decide abortar la operación. El sistema no adoptara los datos capturados como valores válidos de calibración.

Número: 003

Nombre de Caso de Uso-Diseño: “Manipular el Visor 3D”

Estado: (Liberad/ En Revisión/ En Desarrollo)

Autor: Eivar Rojas

Diagrama de Clases

Ilustración **38**. Caso de Uso-Diseño 03 “Manipular el Visor 3D” (Diagrama de Clases)

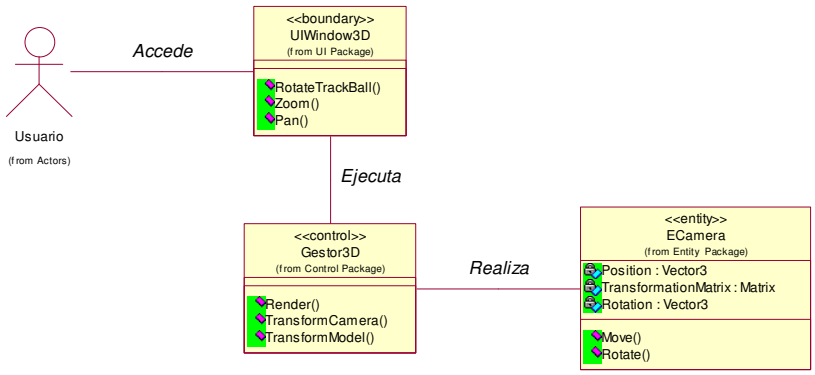
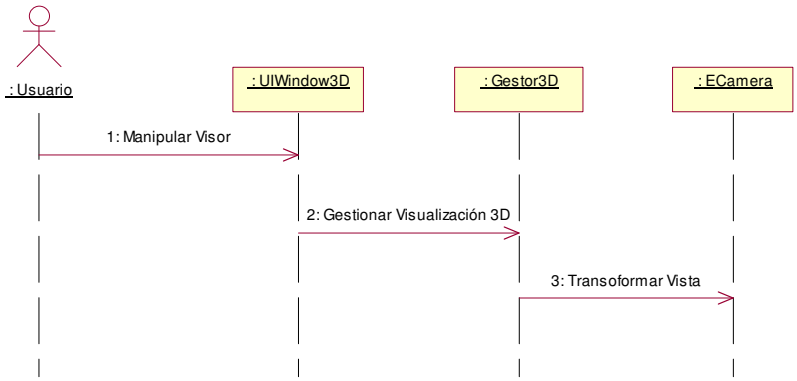


Diagrama de Interacción

Ilustración **39**. Caso de Uso-Diseño 03 “Manipular el Visor 3D” Flujo Alternativo (Diagrama de Interacción)



La ejecución es secuencial en el tiempo, comunicándose con la clase de control Gestor3D con su respectiva clase entidad Camera. El usuario selecciona la opción de manipular el visor 3D. El gestor 3D adoptar y aplica la transformación respectiva a la cámara.

Número: 004

Nombre de Caso de Uso-Diseño: “Reproducir Captura de Movimiento”

Estado: (Liberad/ En Revisión/ En Desarrollo)

Autor: Eivar Rojas

Diagrama de Clases

Ilustración **40**. Caso de Uso-Diseño 04 “Reproducir Captura de Movimiento” (Diagrama de Clases)

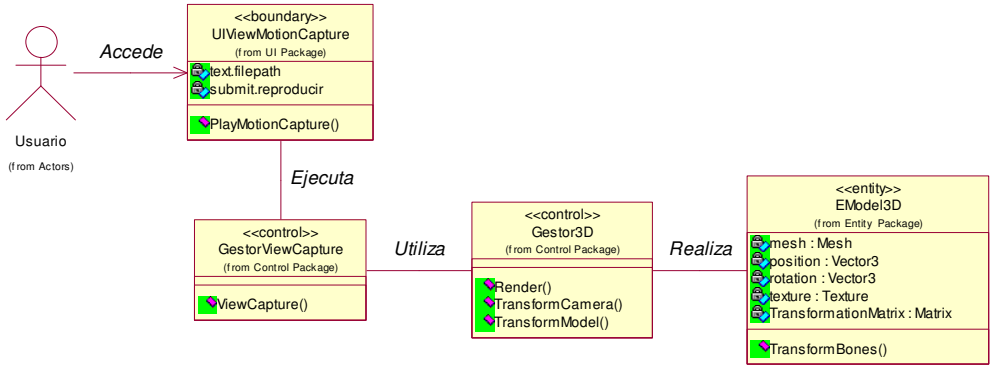
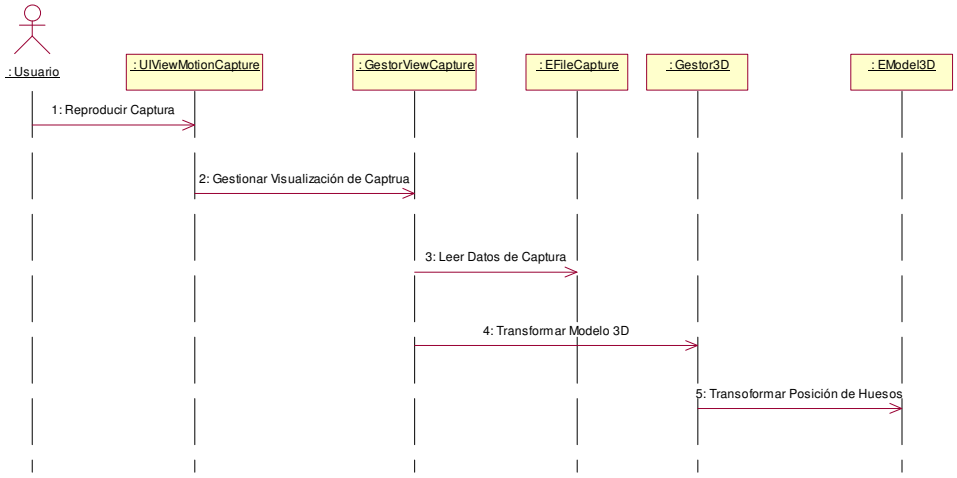


Diagrama de Interacción

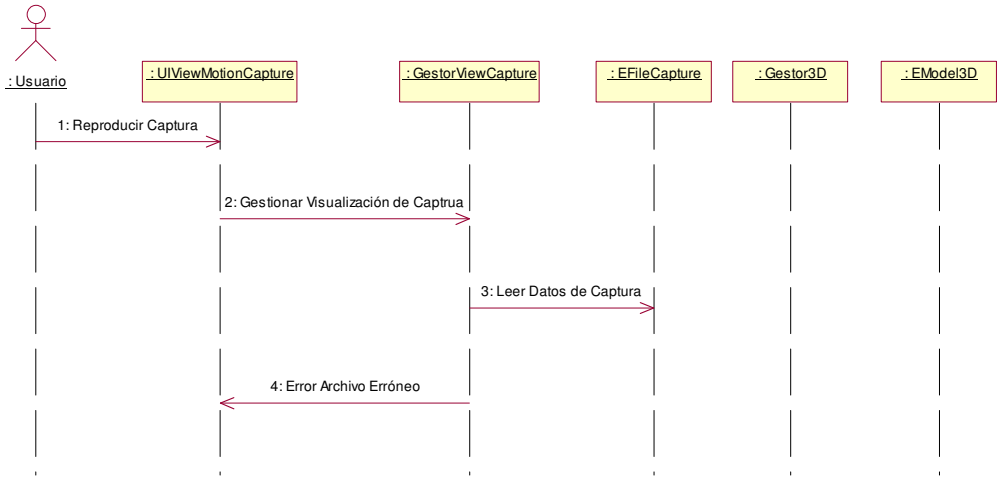
Ilustración **41**. Caso de Uso-Diseño 04 “Reproducir Captura de Movimiento” (Diagrama de Interacción)



La ejecución es secuencial en el tiempo, comunicándose con las clases de control GestorViewCapture y Gestor3D con sus respectiva clases entidad FileCapture y Model3d. El usuario selecciona la opción de visualizar captura especificando el archivo de captura. El gestor de visualización de captura adopta los datos y procede a realizar la operación.

Flujo Alternativo

Ilustración **42**. Caso de Uso-Diseño 04 “Reproducir Captura de Movimiento” Flujo Alternativo (Diagrama de Interacción)



La ejecución es secuencial en el tiempo, comunicándose con las clases de control GestorViewCapture y Gestor3D con sus respectiva clases entidad FileCapture y Model3d. El usuario selecciona la opción de visualizar captura especificando el archivo de captura. Al leer el archivo de captura se encuentra errores en el archivo y se aborta la operación.

Anexo 5. Tutorial Direct3D

Objetivos

- Adoptar los conceptos básicos de la generación de gráficos 3D.
- Creación de un dispositivo de Render 3D.
- Renderización de un triángulo.
- Utilización de matrices de transformación y visualización.
- Uso de texturas.
- Visualización del espacio 3D.
- Manipulación de un Modelo jerárquico 3D.

CONCEPTOS BÁSICOS

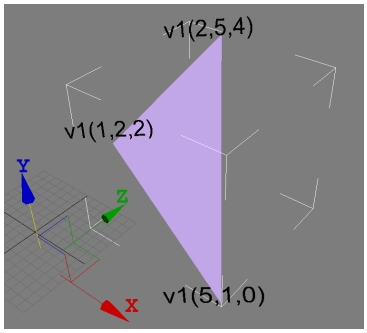
Direct3d utiliza el sistema de coordenadas de la mano izquierda (left handed, OpenGL utiliza el opuesto, right handed). Lo ejes quedan descritos de la siguiente manera:

- Eje X positivo hacia la derecha.
- Eje Y positivo hacia arriba.
- Eje Z se aleja hacia dentro de la pantalla.

Para definir un vértice en el espacio 3D necesitamos especificar sus 3 componentes de coordenadas x, y, z. el origen de las coordenadas es 0, 0, 0.

A continuación se describe tres vértices que conforman un triángulo o polígono:

Ilustración **43**. Vértices que conforman un triángulo (v1 (1, 2, 2).v2 (2, 5, 4).v3 (5, 1, 0).)



Como se puede observar lo anterior un vértice se puede definir por un vector que va desde el origen de coordenadas hasta el punto 3D en el espacio.

Direct3d tiene dos estructuras para definir vectores: D3DVECTOR y D3DXVECTOR3, D3DVECTOR maneja las rutinas matemáticas que se necesita para usar un vector. D3DXVECTOR3 es una versión mejorada de la anterior estructura.

Direct3d utiliza el sistema de coordenadas de la mano izquierda (left hended, OpenGL utiliza el opuesto, right handed).

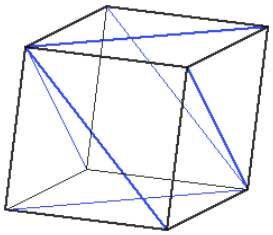
Para definir un vértice en el espacio 3D necesitamos especificar sus 3 componentes de coordenadas x, y, z. el origen de las coordenadas es 0, 0, 0.

CARAS Y NORMALES

Cuando se quiere representar cualquier objeto 3D se necesita recurrir a puntos, líneas y sobre todo triángulos (Polígonos). Por qué triángulos, por que es la unidad mínima para modelar cualquier tipo de objeto complejo, además, por que las tarjetas de video trabajan muy rápidamente haciendo cálculos con triángulos.

A continuación (Ver gráfica) tenemos un cubo de 6 caras compuesto por 12 triángulos:

Ilustración **44**. Composición de un cubo por 12 triángulos



Direct3d solo dibuja las caras frontales de los objetos, que por defecto son aquellas caras que formen los vértices agrupados de acuerdo al sentido de las manecillas del reloj. Este proceso se denomina “Backface Culling” y se basa en eliminar aquellos triángulos que no miran a la cámara por medio de sus normales. En Direct3d se aplican las normales para saber la dirección a la que apunta una cara, o dicho de otra manera, la parte visible de la cara. Normal es el vector unitario perpendicular a la cara. Generalmente podemos definir normales en cada un de los vértices e la cara. En las graficas siguientes se visualiza la normal para una cara y las normales de los vértices:

Ilustración **45**. Normales de una geometría

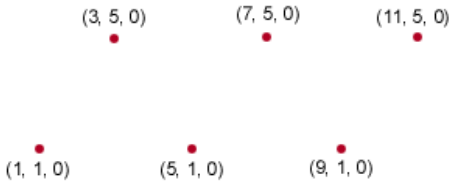
REPRESENTACIÓN

Existen varias formas para que Direct3D interprete un listado de vértices para dibujar un objeto 3D. Son las siguientes:

Ponit List

Lista de puntos (Vértices). Direct3D dibuja puntos en la pantalla los cuales tienen el tamaño de un píxel, independientemente a la distancia a la que se encuentren

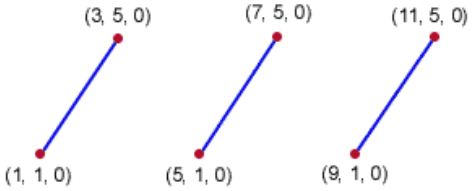
Ilustración **46**. Lista de vértices



Line List

Lista de líneas, cada dos puntos se conforma una línea.

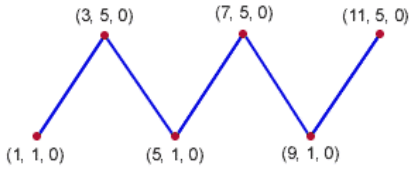
Ilustración **47**. Lista de líneas



Line Strip

“Tira de Líneas”, son líneas unidas entre sí por un solo vértice.

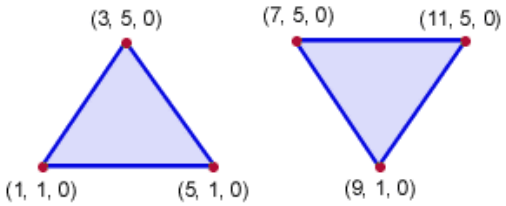
Ilustración **48**. Tira de líneas



Triangle List

Lista de triángulo, cada 3 vértices se forma un triángulo.

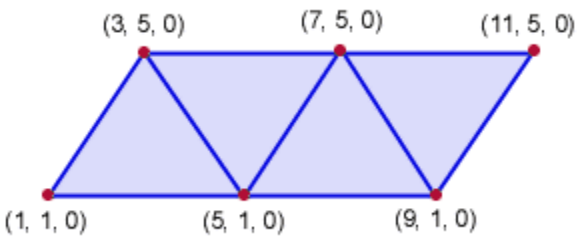
Ilustración **49**. Lista de triángulos



Triangle Strip

“Tira de triángulos, cada triángulo comparte dos vértices para genera el triángulo siguiente. De esta forma se ahorra la cantidad de vértices en lista.

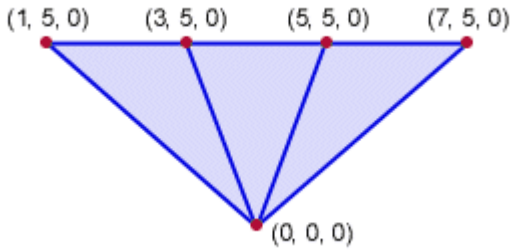
Ilustración **50**. Tira de triángulos



Triangle Fan

Conjunto de triángulos que tienen un vértice en común. Uno de los mejores métodos para generar objetos 3D.

Ilustración **51**. Abanico de triángulos



ARQUITECTURA DE VISUALIZACIÓN

Una de las partes importantes del Pipeline de Direct3D es la parte de transformación (Transformation & Lighting o Vertex Shaders).

Básicamente consiste en operaciones que realizamos para transformar los valores de nuestro mundo 3D entre los diferentes espacios que lo componen. Se descompone el mundo 3D en diferentes sistemas de coordenadas más sencillos para trabajar con él, ya que sería demasiado complicado trabajar sobre él. Estos diferentes sistemas se denominan “espacios”. Estos “espacios” son: espacio-modelo, espacio-mundo, espacio-vista y espacio-proyección.

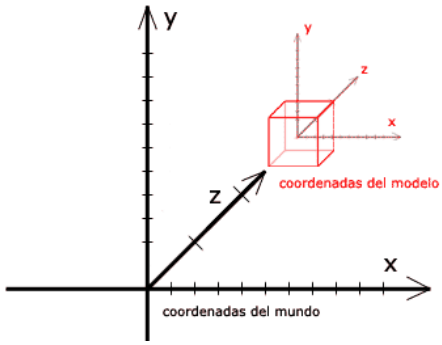
Ilustración **52**. Pipeline Direct3D

El Pipeline transforma cada vértice de un objeto, desde un punto abstracto en el sistema de coordenadas, aun píxel de la pantalla. Teniendo en cuenta las propiedades de la cámara virtual desde la que se ve la escena. Para realizar transformaciones necesitamos 3 matrices que definan nuestros “espacios”. La Matriz-Mundo, La Matriz-Vista y la Matriz-Proyección.

El primero de los pasos, “Transformar Mundo” lo que hace es transformar un objeto desde el espacio-modelo al espacio-mundo. “Espacio-Modelo” es el sistema de coordenadas en el que definimos el objeto, sin tener en cuenta nada mas. En este modelo podemos rotar el objeto, cambiar de tamaño o trasladarlo de lugar para crear animaciones.

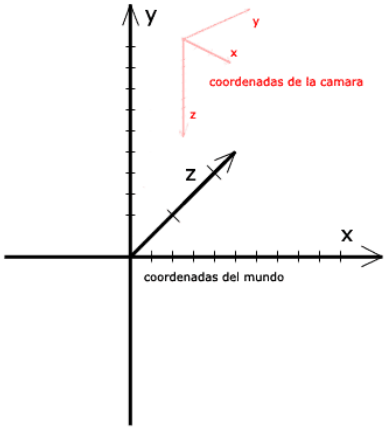
Por ejemplo, si se tiene un personaje que gira el torso, es más fácil calcular nosotros mentalmente los puntos para rotar cuando estamos en el espacio-modelo (el torso en el origen de coordenadas), que cuando estamos en el modelo-mundo (el modelo ubicado en cualquier posición del mundo).

Ilustración **53**. Coordenadas de transformación de modelo



El segundo de los pasos, “Transformar Vista” transforma los objetos del espacio mundo al espacio-vista (o cámara). En este punto podemos decir que tenemos una vista 2D de lo que aproximadamente la cámara ve de nuestro mundo 3D. En este punto también se aplican cálculos de iluminación y de “BackFace culling” (Caras que se muestran y las que están ocultas).

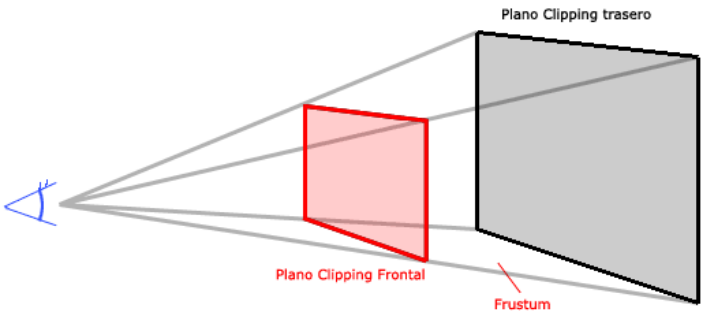
Ilustración **54**. Coordenadas de transformación de cámara



Por último, “Transformar Proyección”, que según sean los campos de vista horizontal y vertical de nuestra cámara crea una adaptación de nuestro modelo-mundo que podemos representar en el monitor “píxel-a-píxel”.

Los objetos que están más cerca de la parte frontal se expanden (“Frustum”), y los que están más lejos se encogen. Esto es posible a través de una matriz de proyección construida a partir del campo de visión (FOV, Field of View) o también llamado “Viewing Frustum”, una aspect ratio, un plano de corte frontal(o Frontal Clipping Plane), y un plano de corte trasero (Back Clipping Plane).

Ilustración **55**. Planos de corte



INICIALIZACIÓN DE DIRECT3D

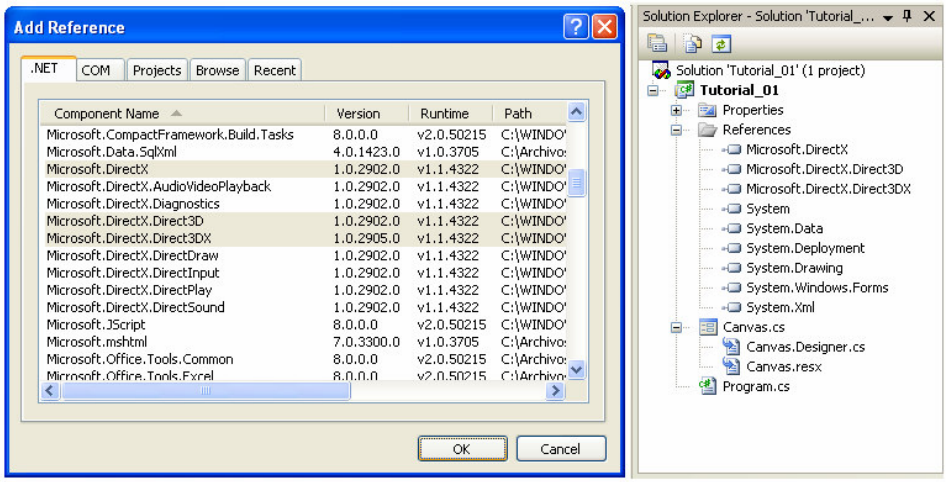
A continuación se construirá una ventana que inicialice D3D, cargando una ventana y que su fondo sea pintado de un color sólido.

La librería a utilizar será Directx 9.0, esta librería esta compuesta de DirectGraphics (o Direct3D), DirectInput, DirectMusic, DirectPlay, DirectShow, DirectSetup, que son un conjunto de interfaces para manejar toda la multimedia del PC.

Para usar estas interfaces se necesita instalar el SDK de DirectX 9.0 y cargar al proyecto las referencias de las librerías.

Iniciamos un nuevo proyecto y en el menú Project se selecciona adicionar Referencia y elegimos Microsoft.Directx, Microsoft.Direct3D y Microsoft.Direct3DX. Se invocan las referencias en las clases donde se requieran objetos de Direct3D.

Ilustración **56**. Librerías de referencia.




```
using Microsoft.DirectX;
using Direct3D=Microsoft.DirectX.Direct3D;
```

El primer objeto que se declara es el dispositivo 3D. Device es el objeto que representa la tarjeta gráfica (Device = Dispositivo). Este objeto es muy importante ya que a través de él se accede a la tarjeta aceleradora, permitiendo crear transformaciones, dibujar triángulos, texturizar y cualquier otro procedimiento que permita manipular el mundo 3D virtual.

Se inicia creando la función INICIALIZARD3D donde se creará el dispositivo de render. En la función se declara una variable que es una estructura del tipo D3DPRESENT_PARAMETERS donde se especifica las características de la ventana de visualización, donde se indica si se quiere backbuffer, especificamos las dimensiones de la pantalla, se indica si la aplicación corre en modo ventana o pantalla completa, se escoge el intercambio de del backbuffer al frontbuffer que sea mas óptimo para la tarjeta aceleradora y por último se establece que el backbuffer tenga el mismo formato (profanidad de color) que la pantalla tiene actualmente.

```
public bool  InicializarD3D()
      {
         try
         {
            g_pD3D.PresentParameters d3dpp = new  g_pD3D.PresentParameters();

            d3dpp.BackBufferCount = 1;
            d3dpp.BackBufferHeight = this.Height;
            d3dpp.BackBufferWidth = this.Width;
            d3dpp.Windowed = true;
            d3dpp.SwapEffect = g_pD3D.SwapEffect.Discard;
            d3dpp.BackBufferFormat = g_pD3D.Format.A8R8G8B8;

g_pD3DDevice = new g_pD3D.Device(0, g_pD3D.DeviceType.Hardware,this.Handle,
g_pD3D.CreateFlags.HardwareVertexProcessing, d3dpp);

         }
         catch(DirectXException)
         {
MessageBox.Show("No se pudo crear el dispositivo D3D",  "Error",MessageBoxButtons.OK,MessageBoxIcon.Exclamation |
MessageBoxIcon.Error   );
            return false;
         }

         return true;
      }
```

Por último la función que crea el dispositivo D3D que es una instancia de Direct3DX. A esta función se le pasan 5 parámetros:

Se indica que coja el dispositivo de visualización primario de Windows.
Con el segundo parámetro se le indica al D3D que el dispositivo anterior debe ser un dispositivo que soporte una interfaz HAL (Hardware Abstraction Level), es decir, que sea un dispositivo que soporte aceleración hardware a través de D3D.
El siguiente parámetro es un apuntador a la ventana principal de render.
El cuarto parámetro le indica a Direct3D que las operaciones con los vértices se realizarán por hardware.

El último parámetro es la estructura a la que se le asignaron características del dispositivo de visualización.

Por último una función que “limpie” todo lo que se ha creado en la aplicación.

```
public void FnalizarD3D()
    {
        if (g_pD3DDevice != null)
            g_pD3DDevice = null;

    }
```

A continuación se describe la función Render. Esta es una de las funciones más importantes ya que se encarga de dibujar la escena (Renderizar).

```
public void Render(object sender, EventArgs e)
    {
        if (g_pD3DDevice == null)
            return;

g_pD3DDevice.Clear(g_pD3D.ClearFlags.Target  , System.Drawing.Color.Orange.ToArgb(), 1.0f, 0);

        g_pD3DDevice.BeginScene();

        g_pD3DDevice.EndScene();

        g_pD3DDevice.Present();
}
```

La función comienza comprobando si el dispositivo D3D esta inicializado. La siguiente función tiene como objetivo limpiar el buffer de pantalla. Los parámetros de la función Clear son:

Flags que indican que tipo de superficie se limpiará. En este caso se limpiará el buffer de pantalla (Target) y el ZBuffer.

El siguiente parámetro es el color con el cual se limpia la pantalla. En este caso usa el color naranja. Para especificar el color se puede hacer de varias formas.

Una es hacerlo directamente indicando el color en hexadecimal (0xFFFF7F00). La otra y más conocida es utilizando los parámetros R (Red), G (Green) y B (Blue).
El penúltimo parámetro es el valor a poner en el ZBuffer al limpiarlo. Va de 0.0 (posición mas cercana) a 1.0 (posición más lejana).
Por último el valor con el cual se rellenara el buffer estarcido al limpiarlo.

La función BEGINSCENE miembro de g_pD3DDev indica a D3D que a partir de ese momento comenzarán las funciones de render. Después de esta función se pueden colocar todas las funciones que dibujen triángulos, etc. De igual forma como indicamos cuando inicia también se especifica cuando finaliza el proceso de render. Para ello se utiliza la función ENDSCENE que también es miembro de g_pD3DDev.

Por último la función PRESENT que pasa el contenido del BackBuffer al FrontBuffer, es decir, a la pantalla.

Ilustración **57**. Creación y dibujado en el canvas.

DIBUJADO DE UN TRIÁNGULO

Tomando como base el código anterior, se dibujara un triángulo con Direct3D y que esté gestionado por el hardware. Su utilizará un búfer de vértices (Vertex Buffer) y la definición de grupos de vértices.

En primer lugar el tipo de vértice a utilizar es un modelo que tiene coordenadas x, y, z y rhw (rhw indica que se utilizan coordenadas de pantalla y no tridimensionales, así en 800x600 el eje x irá de 0 a 799 y el eje e irá de 0 a 599. Esto se denomina coordenadas transformadas) y un color de vértice.

Se declara una en las variables globales un nuevo objeto LPDIRECT3DVERTEXBUFFER que será el búfer de vértices, la función IniVertexBuffer se encargará de crear y llenar el buffer de vértices.

```
public bool IniVertexBuffer()
{
   try
    {
g_pVB = new g_pD3D.VertexBuffer(typeof(g_pD3D.CustomVertex.TransformedColored), 3, g_pD3DDevice,
g_pD3D.Usage.WriteOnly, g_pD3D.CustomVertex.TransformedColored.Format, g_pD3D.Pool.Default);
g_pD3D.CustomVertex.TransformedColored [] verts = (g_pD3D.CustomVertex.TransformedColored [])g_pVB.Lock(0, 0);

verts[0].X = 320.0f; verts[0].Y = 140.0f; verts[0].Z = 0.5f; verts[0].Rhw = 1.0f;
verts[0].Color = System.Drawing.Color.Blue.ToArgb();

verts[1].X = 420.0f; verts[1].Y = 340.0f; verts[1].Z = 0.5f; verts[1].Rhw = 1.0f;
verts[1].Color = System.Drawing.Color.Green.ToArgb();

verts[2].X = 220.0f; verts[2].Y = 340.0f; verts[2].Z = 0.5f; verts[2].Rhw = 1.0f;
verts[2].Color = System.Drawing.Color.Red.ToArgb();

      g_pVB.Unlock();
      return true;
```

```
    }
    catch (DirectXException)
    {
      return false;
    }
}
```

Después de esto se define el vertex buffer. Para ello se crea una nueva instancia de la clase VERTEXBUFFER miembro de g_pD3D. Esta función tiene los siguientes argumentos:

- Tipo de Vértice que conformará el Vertex Buffer.
- Tamaño de datos que contendrá el buffer. En este caso 3 vértices.
- Dispositivo de Render donde se va a renderizar.
- Tipo de acceso al Vertex Buffer.
- Formato del tipo de vértice.
- Tipo de memoria en la que se almacenarán los vértices. Puede ser memoria de video o memoria del sistema.

Se declara el arreglo 'vértices' del tipo CUSTOMVERTEX.TRANSFORMEDCOLORED, cada uno con su coordenada x, y, z y rhw y un color para cada vértice que en este caso es por este orden: Azul, Verde y Rojo. Estos tres vértices van a definir el triángulo.

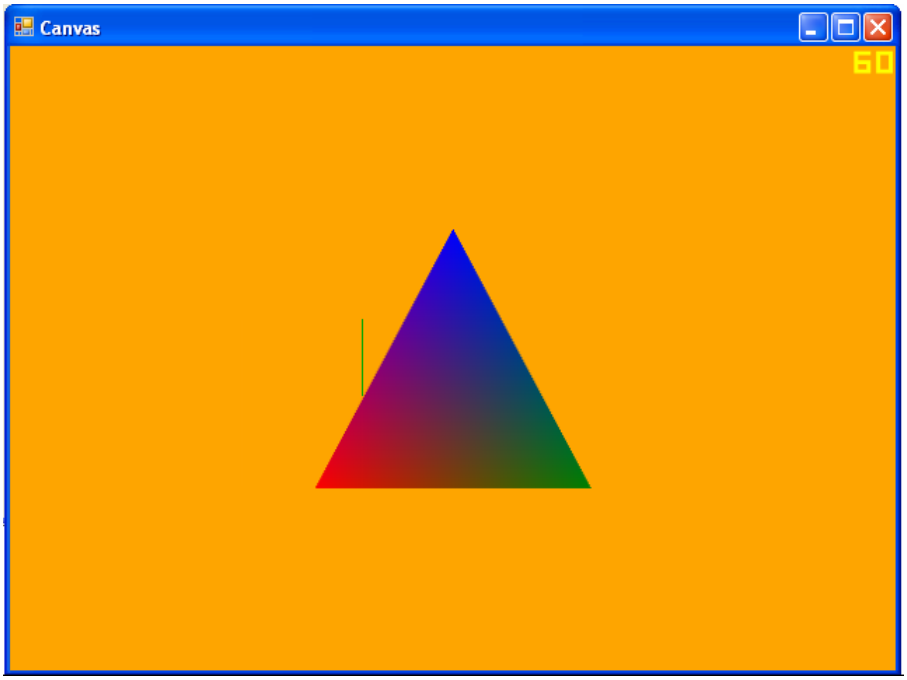
Se bloquea el buffer con Lock. Una vez bloqueado se puede escribir en él. Finalmente desbloqueamos el buffer con Unlock.

Teniendo el buffer de vértices lleno ya se puede dibujar el triángulo. Se modifica la función de Render para dibujar el triángulo.

```
public void Render(object sender, EventArgs e)
    {
      if (g_pD3DDevice == null)
        return;
      g_pD3DDevice.Clear(g_pD3D.ClearFlags.Target, System.Drawing.Color.Orange.ToArgb(), 1.0f, 0);
      g_pD3DDevice.BeginScene();
      g_pD3DDevice.SetStreamSource (0,g_pVB,0);
      g_pD3DDevice.VertexFormat = g_pD3D.CustomVertex.TransformedColored.Format;
      g_pD3DDevice.DrawPrimitives(g_pD3D.PrimitiveType.TriangleList, 0, 3);
      g_pD3DDevice.EndScene();
      g_pD3DDevice.Present();
    }
```

En esta sección se agrega código entre las funciones BeginScene y EndEscene. La función SetStreamSoruce sirve para especificar la fuente donde se encuentran los vértices que definen la geometría a dibujar. De esta función el parámetro de importancia es donde se indica la fuente de los vértices. VertexFormat establece el formato de los vértices. Finalmente se la función DrawPrimitive con el parámetro TriangleList que indica a D3D que debe interpretar la cadena de vértices como un alista en la cual cada 3 vértices definen un triángulo, el vértice inicial y el número de primitivas a dibujar. En este caso se inicia desde el vértice cero y se dibuja un solo triángulo.

lustración **58**. Dibujado de un triángulo en el canvas

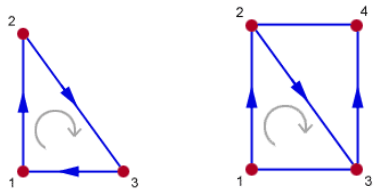


USO DE MATRICES

Tomando como base el tema anterior se añaden más vértices a la escena para crear un cubo y mediante transformaciones por matrices lograr que rote sobre si mismo. La estructura de los vértices pasa de ser coordenadas de pantalla a coordenadas de espacio.

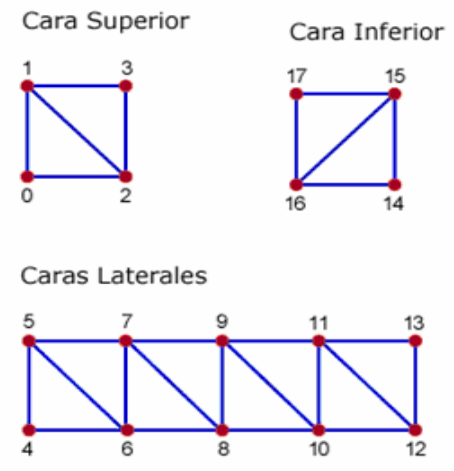
Para generar los triángulos que conforman el cubo deben adoptar un orden especial y siguiendo la dirección de las manecillas del reloj (Clock Wise - CW).

Ilustración **59**. Orden de construcción de un triángulo



El cubo esta compuesto por 3 Triangle Strips. Los cuatro primeros vértices componen el primer strip con 2 triángulos para la cara superior. Las caras laterales es el segundo strip con 10 vértices y un total de 8 triángulos (2 por cada cara).
El tercer strip es la cara inferior que esta formada de la misma manera que la superior.

Ilustración **60**. Orden de construcción de un cubo



Esta es la función que inicia la geometría:

```
  public bool IniVertexBuffer()
      {
       try
        {
         g_pVB = new g_pD3D.VertexBuffer(typeof(g_pD3D.CustomVertex.PositionColored), 18, g_pD3DDevice,
g_pD3D.Usage.WriteOnly, g_pD3D.CustomVertex.PositionColored.Format, g_pD3D.Pool.Default);
         g_pD3D.CustomVertex.PositionColored [] verts = (g_pD3D.CustomVertex.PositionColored [])g_pVB.Lock(0, 0);

            //Cara Superior
         verts[0].Position = new Vector3(-5.0f, 5.0f, -5.0f); verts[0].Color =   Color.FromArgb(0, 0, 255).ToArgb();
         verts[0].Position = new Vector3(-5.0f, 5.0f, 5.0f); verts[0].Color = Color.FromArgb(255, 0, 0).ToArgb();
         verts[0].Position = new Vector3(5.0f, 5.0f, -5.0f); verts[0].Color = Color.FromArgb(255, 0, 0).ToArgb();
         verts[0].Position = new Vector3(5.0f, 5.0f, 5.0f); verts[0].Color = Color.FromArgb(0, 255, 0).ToArgb();
            //Cara 1
         verts[0].Position = new Vector3(-5.0f, -5.0f, -5.0f); verts[0].Color = Color.FromArgb(255, 0, 0).ToArgb();
         verts[0].Position = new Vector3(-5.0f, 5.0f, -5.0f); verts[0].Color = Color.FromArgb(0, 0, 255).ToArgb();
         verts[0].Position = new Vector3(5.0f, -5.0f, -5.0f); verts[0].Color = Color.FromArgb(0, 255, 0).ToArgb();
         verts[0].Position = new Vector3(5.0f, 5.0f, -5.0f); verts[0].Color = Color.FromArgb(255, 0, 0).ToArgb();
            //Cara2
         verts[0].Position = new Vector3(5.0f, -5.0f, 5.0f); verts[0].Color = Color.FromArgb(0, 0, 255).ToArgb();
         verts[0].Position = new Vector3(5.0f, 5.0f, 5.0f); verts[0].Color = Color.FromArgb(0, 255, 0).ToArgb();
```

```
            //Cara3
        verts[0].Position = new Vector3(-5.0f, -5.0f, 5.0f); verts[0].Color = Color.FromArgb(0, 255, 0).ToArgb();
        verts[0].Position = new Vector3(-5.0f, 5.0f, 5.0f); verts[0].Color = Color.FromArgb(255, 0, 0).ToArgb();
            //Cara4
        verts[0].Position = new Vector3(-5.0f, -5.0f, -5.0f); verts[0].Color = Color.FromArgb(255, 0, 0).ToArgb();
        verts[0].Position = new Vector3(-5.0f, 5.0f, -5.0f); verts[0].Color = Color.FromArgb(0, 0, 255).ToArgb();

            //Cara inferior
        verts[0].Position = new Vector3(5.0f, -5.0f, -5.0f); verts[0].Color = Color.FromArgb(0, 255, 0).ToArgb();
        verts[0].Position = new Vector3(5.0f, -5.0f, 5.0f); verts[0].Color = Color.FromArgb(0, 0, 255).ToArgb();
        verts[0].Position = new Vector3(-5.0f, -5.0f, -5.0f); verts[0].Color = Color.FromArgb(255, 0, 0).ToArgb();
        verts[0].Position = new Vector3(-5.0f, -5.0f, 5.0f); verts[0].Color = Color.FromArgb(0, 255, 0).ToArgb();

g_pVB.Unlock();
            return true;
         }
         catch (DirectXException)
         {
            return false;
         }

      }
```

Ya construido el Búfer de vértices ahora solo queda renderizarlo. Para visualizarlo utilizaremos la función ActualizarMatrices que se encarga de crear las matrices necesarias para la visualización del objeto.

Para entender el concepto de esta función vamos a tomar como base un triángulo. Este triángulo se define por tres vértices respecto al origen de coordenadas y a esto se le denomina coordenadas del objeto. Si queremos mover el triángulo a otra posición en un mundo 3D. Se podría pensar que esto se hace modificando las coordenadas de sus vértices y realmente esto se puede hacer pero consumiría demasiado tiempo de proceso. En lugar de eso, se define una matriz de transformación (en este caso traslación) la cual será multiplicada por todos los vértices del triángulo y hará que el triángulo se mueva a la nueva posición. A estas

coordenadas transformadas del triángulo se le denominan coordenadas de mundo.

Debido a que la pantalla es un plano 2D y no puede representar directamente el triángulo ya que este tiene una tercera dimensión. Esto obliga a que se tenga que utilizar otro tipo de transformación que es la proyección. Hay varios tipos de proyección pero las más usadas son la proyección ortográfica o isométrica y la proyección cónica. En cualquiera de los dos casos de proyección, su misión es proyectar en un plano bidimensional un objeto tridimensional de forma que esté preparado para ser plasmado en la pantalla. El plano de la pantalla en el cual se proyecta el objeto 3D es el plano posterior del frustum. El frustum es una figura geométrica la cual es una pirámide cuya cúspide ha sido recortada. El lugar donde se encontraba la cúspide es donde se sitúa la cámara, que en definitiva es el punto de vista desde donde se observa el objeto. Según como se defina el frustum, así se proyectará el objeto 3D. Si el frustum está definido para una proyección ortogonal se tiene que las dimensiones del objeto siempre son las mismas. Por ejemplo, si existen dos objetos con las mismas dimensiones, aunque uno esté muy cerca de la cámara y el otro muy lejos, se observa ambos objetos con el mismo tamaño. Sin embargo la proyección cónica respeta la perspectiva de los objetos.

Por último se debe especificar el punto de vista para ver la escena, definiendo una matriz que es la matriz de vista. Con ella se especifican las coordenadas en las que se encuentra situada la cámara y un vector director que indica hacia donde mira la cámara.

Se inicia definiendo las matrices de rotación del tipo de objeto D3DXMATRIX que representa matrices en D3D. Ahora se asignan los datos necesarios para que se

conviertan en matrices de rotación en el eje X, Y Z respectivamente. El procedimiento para realizar esta operación es con la función Matrix.RotationY. Esta función toma el parámetro ángulo especificado en radianes.

Después de crear las matrices, se multiplican para anularlas en una sola (matWorld). Esto se hace con la función Matrix.Multiply.

```
//Especificiación de matrices de rotación en el eje x,y,z para la matriz mundo
        //esto hará que rote al rededor de los ejes del mundo 3D

        Matrix matWorld, matWorldX, matWorldY, matWorldZ;
        int iTime = Environment.TickCount;
            float fAngle = iTime  / 400.0f;

        //Consturccion de las matrices de transformación
        matWorldX = Matrix.RotationX(fAngle);
        matWorldY = Matrix.RotationY(fAngle);
        matWorldZ = Matrix.RotationZ(fAngle);

        //Combinación de matrices
        matWorld = Matrix.Multiply(matWorldX, matWorldY);
        matWorld = Matrix.Multiply(matWorld, matWorldZ);
```

Teniendo la matriz final de rotación en matWorld no es suficiente. Ya que si se ejecutara la aplicación el triángulo no rotaría, debido a que no se ha especificado la matriz de rotación como la nueva matriz de mundo. Esto se hace con la función SetTransform miembro del dispositivo g_pD3DDev. Esta función toma los siguientes parámetros.

El primero de ellos especifica que matriz de las que tiene D3D se va a modificar. Estos parámetros pueden ser:

- View, encargada de mover el mundo para que sea visto desde el punto que se especifique.
- World, multiplica todos los vértices de los objetos que se rendericen.
- Projection, especifica la matriz de proyección.
- Texture 0… Texture 7, que realiza transformaciones con las coordenadas de textura.

El segundo parámetro es la matriz que contiene la información adecuada para que se generen los efectos deseados.

```
//Se aplica la transformación
    g_pD3DDevice.SetTransform(g_pD3D.TransformType.World, matWorld);
```

Ahora se especifica la matriz de vista. Con esta matriz de vista se logra ubicar la cámara a través de la cual veremos el mundo 3D en la posición y dirección adecuada.

Se crea la matriz de vista con la función D3DXMatrixLookAtLH. Esta función crea una matriz de vista basado en el sistema de coordenadas cartesiano de la mano izquierda (LH = Left handed). El otro modo es el de la mano derecha (RH = right handed). Direct3D se basa en el sistema de coordenadas cartesiano de la mano izquierda mientras OpenGl se basa en el de la mano derecha. Esto significa que en D3D la Z positiva va hacia dentro del monitor y en OGL al revés. Los parámetros de la función que crea la matriz de vista son:

- Un vector del tipo D3DXVECTOR3 que indicará la posición de la cámara.
- Un vector que indica hacia donde mira la cámara, es decir, el vector dirección.

- Un Vector que indica hacia donde está arriba.

El primer parámetro no genera duda. Pero, el segundo y el tercero si. El vector "Up" (arriba) es el resultado del producto vectorial entre el vector "Right" (derecha) y el vector "LookAt" (mirar hacia). El vector "Right" es el resultado del producto vectorial entre los vectores "Up" y "LookAt". El vector "LookAt" se ha especificado que mire hacia el origen de coordenadas que es donde se encuentra el cubo. Si se hubiera querido que la cámara mirara hacia la izquierda, se debería especificar un vector tal como (-1.0f, 0.0f, 0.0f). Es importante saber que los vectores "Up" y "LookAt" debe estar normalizados, es decir, cualquiera de sus componentes debe ser 1.

Después de construir la matriz vista se procede a llamar la función SetTransform con los parámetros adecuados.

```
Matrix matView;

matView = Matrix.LookAtLH(new Vector3(0.0f, 0.0f, -30.0f), new Vector3(0.0f, 0.0f, 0.0f), new   Vector3(0.0f, 1.0f, 0.0f));
        g_pD3DDevice.SetTransform(g_pD3D.TransformType.View, matView);
```

La última matriz a configurar es la matriz de proyección. D3DXMatrixPerspectiveFovLH. Como en la matriz de vista, la proyección se ve condicionada por el sistema de coordenadas mano izquierda o mano derecha. Es este caso se crea una matriz de proyección en perspectiva de mano izquierda. Los parámetros de esta función son:

- FOV (field of view = campo de visión). El FOV es un ángulo que se especifica en radianes.
- Aspect Ratio que es la relación entre el ancho y el alto de la ventana.

- Near Plane (Plano cercano). Es el plano que ha mas cerca de la cámara y todo objeto que quede por detrás de él no será visible.
- Far Plane (Plano Lejano). Es el plano más alejado de la cámara y todo objeto que esté mas allá de él no será visible.

Tanto el “near plane” como el “far plane” forman parte del frustum. El frustum esta formado por 6 planos y todo aquello que este totalmente fuera de él no será visible.

```
Matrix matProj;

matProj = Matrix.PerspectiveFovLH((float)System.Math.PI / 4,(float) this.Width / this.Height,1.0f, 500.0f);
        g_pD3DDevice.SetTransform(Microsoft.DirectX.Direct3D.TransformType.Projection, matProj);
```

Por último en la función INICIALIZARD3D se añade las siguientes líneas:

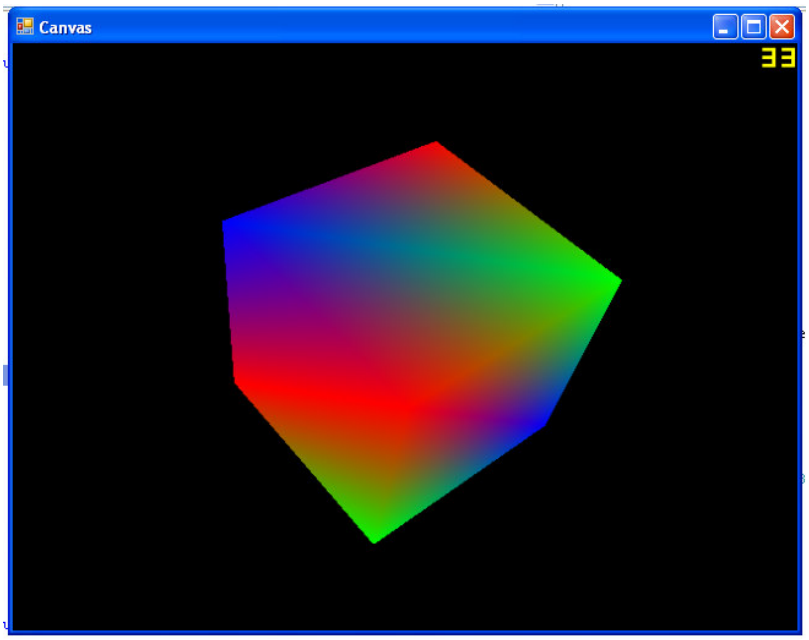
```
g_pD3DDevice.RenderState.CullMode = g_pD3D.Cull.CounterClockwise;
 g_pD3DDevice.RenderState.Lighting = false;
```

La primera activa el Back Face Culling. El culling es la técnica de ocultación de caras posteriores. Hay 3 opciones que son: D3DCULL_CW, D3DCULL_CCW Y D3DCULL_NONE. Con la primera opción se le indica a D3D que las caras de los objetos que se quieren ocultar están formadas por triángulos cuyos vértices han sido especificados en el sentido de las agujas del reloj (CW = clock wise). Con el segundo método es todo lo contrario. Los vértices que se quieren ocultar se han especificado en sentido contrario al as agujas del reloj (CC = counter-clockwise) y por último, D3DCULL_NONE, hacer caso omiso al orden de los vértices renderice ambas caras del triángulo. Todo esto es muy importante por que si por ejemplo se crea un objeto con triángulos creados en sentido CW y se especifica a D3D un culling tipo CW, el resultado es que no se va a ver el objeto. Ya que las caras que están siendo renderizadas son las caras posteriores al as que se esperan ver. En

un objeto convexo serían las que dan al interior del objeto (si se entrara en el objeto, se verían las caras). Solo cuando se le establezca a D3D que debe tomar triángulos a ocultar como CCW se renderizarían bien los triángulos que conforman el objeto. Es muy conveniente que nunca se desactive el culling por razones de optimización.

La segunda desactiva la iluminación. Debido a que no se quiere que D3D calcule el color difuso. Si se hubiese activado la iluminación, la componente difusa la hubiese proporcionado la fuente de luz y con esta información D3D, a través de cálculos de interpolación del color entre vértices y degradación. Si se activara la iluminación el cubo no se vería por dos razones. La primera es que no se ha especificado una fuente de luz y la segunda es que no se ha especificado ninguna normal a los vértices. Una norma es un vector perpendicular a una superficie.

Ilustración **61**. Dibujado de un cubo en el Canvas



USO DE TEXTURAS

Para añadir una textura al cubo. Lo primero que se debe agregar es la variable de tipo LPDIRECT3DTECTURE9, que va a ser la textura:

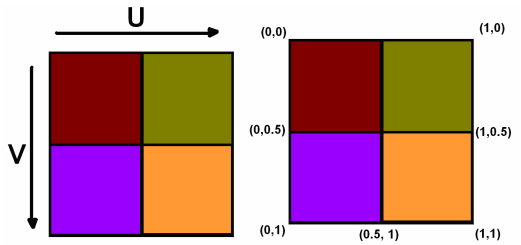
```
g_pD3D.Texture m_pTextura = null;
```

Además, la estructura del tipo de vértice cambia, debido a que es necesario que cada vértice contenga dos nuevas variables (tu y tv) que sirven para mapear la textura. Las coordenadas de textura se asignan directamente a los vértices. De esta manera se puede controlar que porción de la textura es mapeada en la primitiva. Por ejemplo, si se ubica un vértice de la esquina superior izquierda de un cuadrado la posición tu, tv (0.0f, 0.0f) y en le vértice inferior derecho ponemos (1.0f, 1.0f). La textura se mapeará entera sobre el cuadrado. También se tiene que indicar el vértice superior derecho (1.0f, 0.0f) y en el vértice inferior izquierdo (0.0f, 1.0f). En cambio si en lugar de poner 1.0f se coloca 0.5f se mapearía la mitad. 0.0 indica comienzo de la textura, ya sea para arriba de ésta o por su izquierda y 1.0 indica su fin. Si se coloca un valor superior a 1 la textura se repetiría al mapearse en la primitiva.

Si se tiene una textura (textura.bmp) que esta dividida en su interior en 4 cuadrados del mismo tamaño que contienen un dibujo distinto cada uno, y si se quisiera coger el cuadrado que se encuentra en la parte inferior derecha para texturizar. Se tendría que poner el vértice de la esquina superior izquierda de la primitiva (0.5f, 0.5f) por que se inicia en la mitad de la textura, tanto como X como Y (en texturas se llama U y V de ahí tu y tv). En la esquina inferior derecha (1.0f, 1.0f) ya que se tiene el final de la textura. En la superior derecha (1.0f, 0.5f) por

que en U se termina la textura pero en V se encuentra en la mitad. Y en la inferior izquierda (0.5f, 1.0f).

Ilustración **62**. Coordenadas de mapa



Una vez mapeada la textura la variable m_pTextura se encargara de cargar y almacenar la textura.

Ilustración **63**. Ejemplo de textura

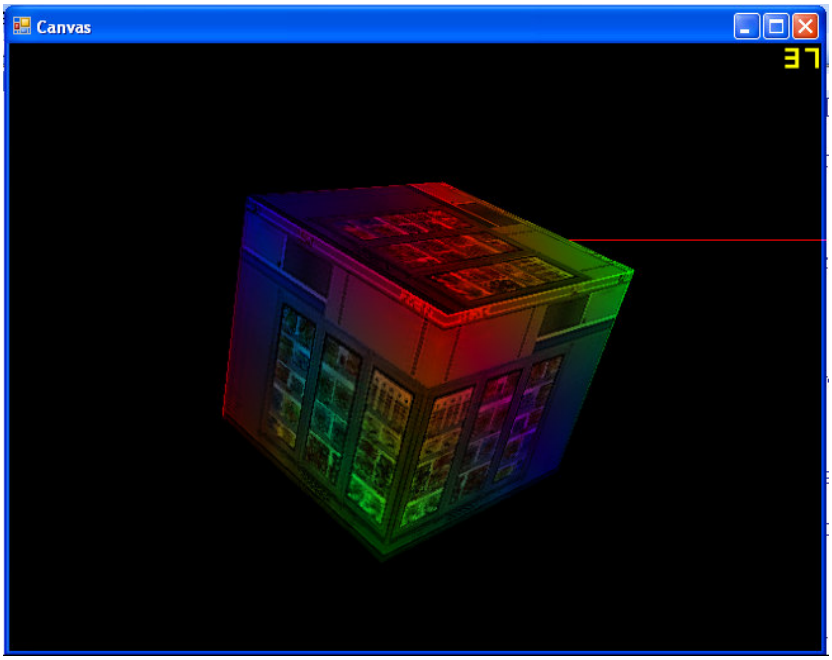
Con la función D3DXCREATETEXTUREFROMFILE miembro de D3D se realizará la carga del archivo de imagen (BMP, TGA Y JPEG) en memoria. Ahora solo queda renderizar el cubo con su respectiva textura.

Si la textura ha sido inicializada se va a cargar para renderizar con SetTexture que toma como argumentos:

- El número de stage al cual se le asignará dicha textura.
- La textura.

En Direct3D hay un máximo de 8 stages. Los stages son zonas en las que se almacenan unas determinadas texturas (con sus propiedades) para posteriormente realizar algún tipo de operación entre ellas. Normalmente siempre se utiliza el stage 0, debido, a que no ser realizara ningún tipo de transformación a la textura.

Ilustración **64**. Dibujado de un cubo con textura



CÁMARA EN ESPACIO

Cuando se piensa en una cámara se piensa en un objeto a través del cual podemos ver el mundo que nos rodea desde un punto de vista determinado y con un campo de visión específico. Básicamente en D3D (y cualquier API 3D) la cámara es representa lo mismo, es decir una "herramienta" que permite posicionarnos en la situación más ventajosa del mundo 3d. Pero la similitud entre una cámara "real" y una cámara "Virtual" termina ahí. Por ejemplo, si se quisiera rotar la cámara "real" 90º a la derecha tan solo hay que rotarla 90º a la derecha. Pero si se quiere rotar la cámara en D3D a la derecha lo que se tiene que hacer es rotar todo el mundo 3D 90º a la izquierda. De igual forma, si se quiere trasladar la cámara 10 unidades hacia delante lo que realmente debe suceder es que todo el mundo se traslada -10 unidades en el eje Z.

Para lograr aplicar transformaciones a la cámara lo que realmente sucede es que en realidad la cámara se encuentra siempre en el origen de coordenadas y que todo debe rotarse y trasladarse en relación a ella. Las traslaciones y rotaciones se consiguen con matrices. Debido a que las matrices no cumplen la propiedad conmutativa, es decir, que no es lo mismo multiplicar la matriz A por la matriz B que por la matriz B por la matriz A. Es por eso que se debe crear una matriz de vista (la matriz que posiciona el punto de vista) regida por la siguiente fórmula:

```
ViewMatrix = TranslationMatrix * RotationYMatrix * RotationXMatrix * RotationZMatrix
```

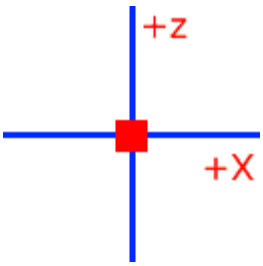
Como se dijo anteriormente para rotar la cámara se debe hacer lo mismo con el mundo pero en sentido inverso y lo mismo con la traslación. También, se debe rotar el mundo en relación a la cámara, es decir, que el eje de rotación del mundo es la posición de la cámara o el origen del eje de coordenadas. Para rotar el

mundo alrededor de la cámara primero se debe trasladar a la posición de la cámara pero invertido y rotarlo la cantidad necesaria de forma inversa. Es por eso que la concatenación de las matrices de transformación de mundo se presenta de la siguiente forma (Primero se rota el objeto y luego se traslada):

```
WorldMatrix = RotationYMatrix * RotationXMatrix * RotationZMatrix * TranslationMatrix
```

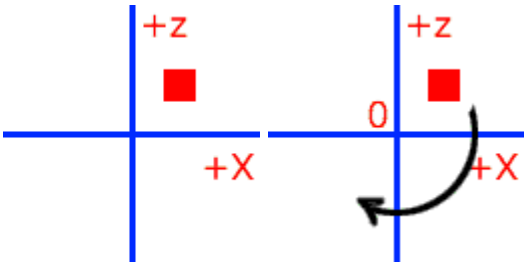
Por ejemplo, si se quiere trasladar un objeto a la posición X = 10 y Y=0 y Z = 10 y se quiere que el objeto rote sobre si mismo en el eje Y 180º. Si se observara el eje de coordenadas desde arriba se vería lo siguiente:

Ilustración **65**. Eje de coordenadas locales



Ahora se aplica la transformación de forma incorrecta, primero se traslada y luego se rota:

Ilustración **66**. Transformación de la posición y rotación de un objeto de forma incorrecta



Como se puede observar, la rotación no se realiza alrededor del objeto en si, sino que lo hace alrededor del origen de coordenadas, por que para todas las transformaciones se toma como referencia el origen de coordenadas.

Para las rotaciones reciben un nombre especial. A la rotación en el eje X se le conoce como pitch, a la del eje Y se denomina yaw y a la del eje Z se llama roll.

Para la cámara se crea una clase que contenga todas las características necesarias para realizar transformaciones.

```
class CCamara
  {
     float m_fFov;

     int m_iAncho;
     int m_iAlto;

     float m_fPitch;
     float m_fYaw;
     float m_fRoll;

     float m_fPlanoCercano;
     float m_fPlanoLejano;

     Vector3 m_Posicion;

     Vector3  m_Dcha;
     Vector3  m_Arriba;
     Vector3  m_Look;
 }
```

De esta clase se explicará las partes de código más importantes y complejas. Los miembros privados de la clase m_fFov que recibirá el valor del FOV que se le asigne a la cámara, m_iAncho y m_iAlto que especifican el ancho y alto de la pantalla, m_fPtich, m_fYaw y m_fRoll que especifican cada uno de los ángulos de rotación de la cámara. F_PlanoCercano y m_fPlanoLejao que especifican el plano

cercano y el plano lejano del frustum, m_Posicion que contienen la posición de la cámara y finalmente tres vectores; m_Dcha, m_Arriba y m_Look que son los vectores que definen la orientación de la cámara. De estos tres vectores el que más se utiliza es el vector m_Look (LookAt) que es el vector que indica hacia donde apunta la cámara.

De las funciones mas relevantes encontramos ActualizarVista() es donde se implementa la teoría establecida anteriormente.

```
public void ActualizarVista(g_pD3D.Device pD3DDev)
    {
        Matrix T, Rx, Ry, Rz;
        Matrix view;

        // Cogemos el vector dcha, arriba y lookat actualizados
        view = pD3DDev.GetTransform(g_pD3D.TransformType.View);

        m_Dcha = new Vector3 (view.M11, view.M21, view.M31);
        m_Arriba = new Vector3(view.M12, view.M22, view.M32);
        m_Look = new Vector3(view.M13, view.M23, view.M33);

        // Calculamos la nueva vista
        T = Matrix.Translation(-m_Posicion.X, -m_Posicion.Y, -m_Posicion.Z);

        Ry = Matrix.RotationY(-m_fYaw);
        Rx = Matrix.RotationX(-m_fPitch);
        Rz = Matrix.RotationZ(-m_fRoll);

        view = T * Ry * Rx * Rz;

        // La activamos
        pD3DDev.SetTransform(g_pD3D.TransformType.View, view);
    }
```

Lo primero que se hace es actualizar los vectores que definen la orientación de la cámara. Con los tres vectores se arma la matriz vista:

Right.x	Up.x	Look.x	0
Right.y	Up.y	Look.y	0
Right.z	Up.z	Look.z	0
- DotProduct(Position,Right)	- DotProduct(Position,Right)	- DotProduct(Position,Right)	1

En la matriz DotProduct es la operación Producto Escalar entre vectores.

La función Avanzar():

```
public void Avanzar(float fSpeed, float fDTime)
    {
            m_Posicion += fSpeed * fDTime * m_Look;
    }
```

Como se puede observar la posición actual se le suma la velocidad por el tiempo (la posición es igual a la posición inicial más la velocidad por el incremento del tiempo). Pero además se multiplica por el vector m_look. Al multiplicarlo por este vector se escala el valor final pero en el sentido y dirección que especifica dicho vector. Por esta razón se incluye m_look. Si no se especificara ningún vector la cámara no sabría hacia donde moverse.

MANIPULANDO UN MODELO 3D CON HUESOS

El tipo de modelo jeráquico que se va a utilizar es el MD5. Este formato es implementado y creado por la empresa desarrolladora de video juegos ID Software. Existen versiones hacia atrás, esta versión es la última conocida e implementada por ID Software en su último videojuego DoomIII.

Las razones para utilizar este formato de archivo es su amplia divulgación en la Internet debido a la existencia de un amplio grupo de fanáticos de este video juego. Además, la ordenda esxtructura que lo compone bridando una gran ayuda a la generación de modelos poseedores de jerarquías de huesos.

Este modelo contiene la siguiente estructura:

- Versión de compilación
- Región de Huesos.
- Región de Mallas.

Cada region está precedida por un indicador de la cantidad de elementos que la componen:

```
Numbones 3

Bone 0 {
        ……..
```

En la primera region se encuentran los huesos presentes en el modelo, donde se indica el nombre, posición (ubicación del hueso) y matriz de transformacion (matriz de transformación es utilizada para afectar los vértices que estan influenciados por este hueso).

```
Bone 0 {
        Name "Bone01"
        Bindpos -0.419597 0.0678973 0.60738
        Bindmat 0.00487154 0.0 0.999988 -0.999988 0.0 0.00487154 0.0 -1.0 0.0
    }
```

En la region de Mallas se describen las mallas que componen el modelo (un modelo puede estar compuesto de muchas mallas) cada malla está compuesta de los siguientes elementos:

- Shader, ubicación de la textura que tiene asignada la malla.
- Vértices, vértices que conforman la malla.
- Triángulos, los polígonos que agrupan los vértices.
- Weights, los “pesos” que afectan a los vértices.

La estructura de los vértices es un poco especial. No contienen su posición, contienen un índice, las coordenadas de mapeo (tu, tv), un puntero al primer “Peso” que afecta al vértice, y la cantidad de “Pesos” que lo afectan (cuando se accede a los pesos se hace de forma secuencial).

```
Vert 0 1.0 1.0 0 1
```

La estructura de los triángulos indica, el índice del triángulo, y los tres apuntadores a los vértices que lo componen.

```
tri 0 0 2 1
```

La estructura de los “Pesos” es el que mas información contiene. El índice del “peso”, el hueso que lo genera, el grado de afectación y posición.

```
Weight 0 0 1.0 -0.66375 10.9905 9.28267
```

Una vez clara la estructura del formato MD5 se crea la clase CLoadMD5 que será la encargada de leer toda la información del archivo.

La clase encargada de guardar toda la información del modelo y la transformación de los huesos (Tema que será explicado mas adelante) será CModel y estará compuesta de los siguientes elementos:

- CMesh, grupo de mallas que componen el modelo.
- CBone, grupo de huesos que conforman el modelo.

```
public class ModelMd5
    {

        /// <summary>Colección de Huesos.      </summary>
        public ArrayList ListaHuesos;

        /// <summary>Colección de Meshes.      </summary>
        public ArrayList ListaMesh;

/// <summary>Posición del modelo.       </summary>
        public Vector3 Posicion;


        public ModelMd5()
        {
                BonesCol = new ArrayList();
                MeshCol = new ArrayList();

        }
    }
```

La clase CMesh contiene los siguientes elementos:

- Lista de vértices.
- Lista de triángulos.

- Lista de pesos.
- Textura asociada a la malla.
- Un Buffer de vértices.

```
class CMesh
  {

    /// <summary> Colección de Vértices.          </summary>
    public ArrayList ListaVertices = new ArrayList();;

    /// <summary> Coleción de Triángulos.         </summary>
    public ArrayList ListaTriangulos = new ArrayList();;

    /// <summary> Colección de Pesos.             </summary>
    public ArrayList ListaPesos = new ArrayList();;

    /// <summary>Bufer de vértices.     </summary>
    public VertexBuffer vertexBuffer;

    /// <summary>Textura de la malla.  </summary>
    public Texture textura;

  }
```

Para cada objeto presente en CMesh, Huesos, Vértices y triángulos se generará una clase respectiva.

De acuerdo a como se explico anteriormente la información que viene contenida en el archivo MD5 se va a describir cada una de las tres clases que conforman a CMesh:

CBone: esta clase representa los huesos contenidos en el modelo:

```
class CBone
  {
```

```
    /// <summary> Indice del elemento en la Colección de Huesos.    </summary>
    public int Index;

    /// <summary> Nombre del Hueso.  </summary>
    public string Nombre;

    /// <summary> Nombre del Hueso pariente.    </summary>
    public string NombrePariente;

    /// <summary> Matriz de transformación.       </summary>
    public Microsoft.DirectX.Matrix MatrizTransformacion;

    /// <summary> Posicion del hueso. </summary>
    public Vector3 Posicion;

    /// <summary> Rotacion del hueso.             </summary>
    public Vector3 Rotacion;

    /// <summary> Angulo de Rotación.             </summary>
    public float Angulo;
  }
```

CVertex: clase que representa los vértices que conforman el mesh:

```
class CVertex
  {
    /// <summary> Indice del elemento en la Colección de Vértices.   </summary>
    public int Indice = 0;

    /// <summary> Coordenada de Textura U </summary>
    public float Tu = 0;

    /// <summary> Coordenada de Textura V </summary>
    public float Tv = 0;

    /// <summary> Indice del primer Peso </summary>
    public int IndexPeso = 0;

    /// <summary> Cantidad de Pesos que afectan este vértice </summary>
    public int CantidadPesos = 0;
```

```
    /// <summary> Posición del Vértice </summary>
    public Vector3 Posicion = new Vector3();

    /// <summary> Caras que hace parte </summary>
    public ArrayList ListaNormales = new ArrayList();

    /// <summary> Vector Normal </summary>
    public Vector3 Normal = new Vector3();
  }
```

CTriangle: contiene la información para crear un triangulo:

```
class CTriangle
  {
    /// <summary> Indice del elemento en la Colección de Triangulos.          </summary>
    public int Indice = 0;

    /// <summary> Indice del Vértice número 1. </summary>
    public int IndiceVert1 = 0;

    /// <summary> Indice del Vértice número 2. </summary>
    public int IndiceVert2 = 0;

    /// <summary> Indice del Vértice número 3. </summary>
    public int IndiceVert3 = 0;
  }
```

CWeight: clase que describe los huesos que afectan al vértice y cuanta es la influencia:

class CWeight

```
  {
    /// <summary> Indice del Peso </summary>
    public int Indice = 0;

    /// <summary> Indice del Hueso que genera el Peso </summary>
    public int Huesos = 0;

    /// <summary> Fuerza del Peso </summary>
    public float Bias = 0.0f;
```

```
    /// <summary> Posición del Peso </summary>
    public Vector3 Posicion = new Vector3();
}
```

Definidas las clases que componen el modelo, se crea la clase encargada de leer el archivo y almacenarlo en una instancia de la clase CModel.

El MD5 se descompone en dos tipos de archivo el primero y el de interés, el md5mesh que es donde se encuentra todo el modelo 3D y el md5anim donde se encuentran todos los keyframe de animación del modelo. El formato del archivo es texto y totalmente secuencia.

El primer dato que se encuentra es la versión del MD5, la versión con la que se va a trabajar es la número 6.

```
MD5Version 6
```

En la segunda línea encontramos un commandline, que para fines de este proyecto no es de interés, pero, este comando sirve para definir parámetros de compilación para el modelo dentro del motor de juego.

Commandline "mesh maps/fred/e3/lobby/lobby3.mb"

En la tercera línea encontramos la cantidad de huesos que conforman el modelo.

```
Numbones 3
```

Seguido de esto se encuentra las definiciones para cada hueso:

```
bone 0 {
name "Bone01"
bindpos -0.419597 0.0678973 0.60738
```

```
bindmat 0.00487154 0.0 0.999988 -0.999988 0.0 0.00487154 0.0 -1.0 0.0
}
```

Luego de la definición de todos los huesos presentes en el modelo se encuentra una línea donde se especifica la cantidad de meshes que conforman el modelo.

```
Nummeshes 1
```

Y la definición para cada mesh. La descripción de un mesh esta compuesto por lo siguiente:

```
Mesh 0 {
        shader "undefined"

        numverts 54
        vert 0 1.0 1.0 0 1
        . . .

        numtris 48
        tri 0 0 2 1
        . . .

        numweights 71
        weight 0 0 1.0 -0.66375 10.9905 9.28267
}
```

Primero se encuentra la dirección donde se encuentra la textura, segundo la cantidad de vértices presentes en el mesh y la descripción de cada uno, cantidad de triángulos presentes en el mesh y descripción de cada uno, cantidad de “Pesos” presentes en el mesh y la descripción de cada uno.

La descripción de cada elemento (Vértices, Triángulos y Pesos) es de forma continua secuencial, cada dato se encentra separado por un espacio.

Teniendo la información del modelo contenida en la clase ya se puede dar paso a ubicar los vértices en sus posiciones, debido que hasta este punto los vértices no tienen una posición asignada.

Para determinar la posición del vértice es necesario saber que huesos afectan al vértice y cada hueso cuanta fuerza produce sobre el vértice. La función que calcula este proceso es la siguiente:

```
public void BindVertex(ArrayList ListaHuesos)
    {

        int k;

        Vector3 Vec1 = new Vector3();

        Vector3 Vert = new Vector3();

        CWeight W = new CWeight();

        CBone B = new CBone();

        Matrix Mat = Matrix.Identity;

        foreach (CVertex Vert in ListaVertices)
        {
           Vert.Posicion = Vector3.Empty;

           for (k = 0; k < Vert.CantidadPesos; k++)
           {
              W = (CWeight)ListaPesos[Vert.IndicePeso + k];

              B = (CBone)ListaHuesos[W.Huesos];

Vec1.X = (W.Posicion.X * B.MatrizTransformacion.M11) + (W.Posicion.Y * B.MatrizTransformacion.M21)  + (W.Posicion.Z *
B.MatrizTransformacion.M31);

Vec1.Y = (W.Posicion.X * B.MatrizTransformacion.M12) + (W.Posicion.Y * B.MatrizTransformacion.M22)  + (W.Posicion.Z *
B.MatrizTransformacion.M32);
```

```
Vec1.Z = (W.Posicion.X * B.MatrizTransformacion.M13) + (W.Posicion.Y *  B.MatrizTransformacion.M23) + (W.Posicion.Z *
B.MatrizTransformacion.M33);

                Vert.Posicion.X = ((Vec1.X + B.Posicion.X) * W.Bias) + Vert.Posicion.X;

                Vert.Posicion.Y = ((Vec1.Y + B.Posicion.Y) * W.Bias) + Vert.Posicion.Y;

                Vert.Posicion.Z = ((Vec1.Z + B.Posicion.Z) * W.Bias) + Vert.Posicion.Z;

            }// for k

        } // for j

    } // Void
```

Primero, en el orden que se agruparon los vértices uno a uno se determina que pesos están relacionados a este. Con el peso se determina a que hueso que tiene relacionado. Con la matriz de transformación contenida en el hueso transformamos la posición del peso, obteniendo esta posición se acumula junto con la posición del hueso, al resultado se le multiplica la cantidad fuerza ejercida por el peso y por último la suma de vértice. Esta suma es acumulativa debido a que un vértice puede ser afectado por varios pesos y varios huesos.

Cada que se transforme la posición de un hueso se debe efectuar este proceso pero además de este proceso falta algo esencial en la visualización, se debe actualizar el buffer de vértices.

Determinada la posición de cada vértice ya se puede calcular la normal y la escritura del buffer de vértices.

El método para calcular la normal para un vértice inicia determinando en cuantos triángulos es utilizado el vértice, cada triángulo contiene una normal, esta normal se le asigna a cada vértice, después de recorrer todos los triángulos que posee el

modelo, cada vértice tiene una lista de normales que fueron asociadas a el, por último se promedian todas las normales que tiene asociadas el vértice y el vector normal final es normalizado.

```
public void CalculateNormals()
    {
        int i = 0;
        int j = 0;
        Vector3 v12 = new Vector3();
        Vector3 v13 = new Vector3();
        Vector3 v = new Vector3();
        CTriangle Tri = new CTriangle();
        CVertex Vert1 = new CVertex();
        CVertex Vert2 = new CVertex();
        CVertex Vert3 = new CVertex();
        Vector3 vectornormal = new Vector3();

        if (ListaVertices.Count > 1)
        {

            for (i = 0; i < ListaTriangulos.Count; i += 1)
            {
                Tri = (CTriangle)ListaTriangulos[i];
                Vert1 = (CVertex)ListaVertices[Tri.IndiceVert1];
                Vert2 = (CVertex)ListaVertices[Tri.IndiceVert2];
                Vert3 = (CVertex)ListaVertices[Tri.IndiceVert3];
                v12 = Vector3.Subtract(Vert2.Posicion, Vert1.Posicion);
                v13 = Vector3.Subtract(Vert1.Posicion, Vert3.Posicion);
                Vert1.ListaNormales.Add(Vector3.Normalize(Vector3.Cross(v12, v13)));
                Vert2.ListaNormales.Add(Vector3.Normalize(Vector3.Cross(v12, v13)));
                Vert3.ListaNormales.Add(Vector3.Normalize(Vector3.Cross(v12, v13)));
            }
        }

        for (i = 0; i < ListaVertices.Count; i++)
        {
            Vert1 = (CVertex)ListaVertices[i];
            vectornormal = Vector3.Empty;
            for (j = 0; j < Vert1.ListaNormales.Count; j++)
            {
```

```
            vectornormal.Add((Vector3)Vert1.ListaNormales[j]);
        }
        Vert1.ListaNormales = new ArrayList();
        v = Vector3.Normalize(vectornormal);
        Vert1.Normal.X = v.Z;
        Vert1.Normal.Y = v.Y;
        Vert1.Normal.Z = v.X;
    } }
```

Para la escritura del buffer de vértices primero se debe tener en cuenta que el tipo de vértice a utilizar es el POSITIONNORMALTEXTURED, a causa de que el vértice además de poseer una posición, cuenta con una asignación de textura y un vector normal (utilizado para determinar el grado de luminosidad que recibe un vértice). Justo antes de escribir el buffer de vértices se calculan las normales.

```
public void WriteVertexBuffer(ArrayList ListaHuesos)
    {
        int i = 0;
        int j = 0;
        CTriangle Tri = new CTriangle();
        CVertex Vert1 = new CVertex();
        CVertex Vert2 = new CVertex();
        CVertex Vert3 = new CVertex();

        BindVertex(ListaHuesos);

        CalculateNormals();
        if (ListaVertices.Count > 1)
        {
            CustomVertex.PositionNormalTextured[] verts = (CustomVertex.PositionNormalTextured[])vertexBuffer.Lock(0, 0);
            j = 0;
            for (i = 0; i < ListaTriangulos.Count * 3; i = i + 3)
            {
                Tri = (CTriangle)ListaTriangulos[j];

                Vert1 = (CVertex)ListaVertices[Tri.IndiceVert1];
                verts[i].Position = Vert1.Posicion;
                verts[i].Normal = Vert1.Normal;
                Vert2 = (CVertex)ListaVertices[Tri.IndiceVert2];
                verts[i + 1].Position = Vert2.Posicion;
```

```
            verts[i + 1].Normal = Vert2.Normal;
            Vert3 = (CVertex)ListaVertices[Tri.IndiceVert3];
            verts[i + 2].Normal = Vert3.Normal;
            verts[i + 2].Position = Vert3.Posicion;
            verts[i].Tu = Vert1.Tu;
            verts[i].Tv = Vert1.Tv;
            verts[i + 1].Tu = Vert2.Tu;
            verts[i + 1].Tv = Vert2.Tv;
            verts[i + 2].Tu = Vert3.Tu;
            verts[i + 2].Tv = Vert3.Tv;
            j += 1;
        }
        vertexBuffer.Unlock();
    }  }
```

Con buffer de vértices correctamente escrito el procedimiento final es renderizar el modelo. Cabe notar que para generar transformación lo único que se debe hacer es mover los huesos.